FERNANDO REYES CORTÉS JAIME CID MONJARAZ

ARDUINO



APLICACIONES EN ROBÓTICA, MECATRÓNICA E INGENIERÍAS





APLICACIONES EN ROBÓTICA, MECATRÓNICA E INGENIERÍAS



Dogram Code

Accede a Gratis a la Biblioteca Online +300 Libros en PDF https://dogramcode.com/biblioteca

> Únete al canal de Telegram https://t.me/bibliotecagratis_dogramcode Únete al Grupo de Facebook

https://www.facebook.com/groups/librosyrecursoselectricidadyelectronica

FERNANDO REYES CORTÉS JAIME CID MONJARAZ

ARDUINO

APLICACIONES EN ROBÓTICA, MECATRÓNICA E INGENIERÍAS





Director Editorial

Marcelo Grillo Giannetto mgrillo@alfaomega.com.mx

Editor

Francisco Javier Rodríguez Cruz jrodriguez@alfaomega.com.mx

Datos catalográficos

Reyes Cortés, Fernando; Cid Monjaraz, Jaime; Arduino. Aplicaciones en Robótica, Mecatrónica e Ingenierías

Primera Edición

Alfaomega Grupo Editor, S.A. de C.V., México

ISBN: 978-607-622-193-8

Formato: 17 cm X 23 cm Páginas: 468

Arduino. Aplicaciones en Robótica, Mecatrónica e Ingenierías

Fernando Reyes Cortés y Jaime Cid Monjaraz.

Derechos reservados © Alfaomega Grupo Editor, S.A. de C.V., México.

Primera edición: Alfaomega Grupo Editor, México, enero de 2015.

 $\ \, \textcircled{o}$ 2015 Alfa
omega Grupo Editor, S.A. de C.V.

Pitágoras 1139, Col. Del Valle, 03100, México D.F.

Miembro de la Cámara Nacional de la Industria Editorial Mexicana

Registro No. 2317

Pág. Web: http://www.alfaomega.com.mx E-mail: atencionalcliente@alfaomega.com.mx

ISBN: 978-607-622-193-8

Derechos reservados:

Esta obra es propiedad intelectual de sus autores y los derechos de publicación en lengua española han sido legalmente transferidos al editor. Prohibida su reproducción parcial o total por cualquier medio sin permiso por escrito del propietario de los derechos del copyright.

Nota importante:

La información contenida en esta obra tiene un fin exclusivamente didáctico y, por lo tanto, no está previsto su aprovechamiento a nivel profesional o industrial. Las indicaciones técnicas y programas incluidos, han sido elaborados con gran cuidado por los autores y reproducidos bajo estrictas normas de control. ALFAOMEGA GRUPO EDITOR, S.A. de C.V. no será jurídicamente responsable por: errores u omisiones; daños y perjuicios que se pudieran atribuir al uso de la información comprendida en este libro, ni por la utilización indebida que pudiera dársele. El texto fue compuesto por los autores en lenguaje LATEX con PCTEX^{MR} 6.0.

Edición autorizada para venta en todo el mundo.

Impreso en México. Printed in Mexico.

Empresas del grupo:

México: Alfaomega Grupo Editor, S.A. de C.V.-Pitágoras 1139, Col. Del Valle, 03100, México D.F. C.P. 03100. Tel.: (52-55)5575-5022 Fax: (52-55)5575-2420/2490. Sin costo: 01-800-020-4396 E-mail: atencionalcliente@alfaomega.com.mx

Colombia: Alfaomega Colombia S.A. Calle 62 No. 20-46 Barrio San Luis, Bogotá, Colombia Tel.: (57-1)746 0102 / 210 0415 E-mail: cliente@alfaomega.com.co

Chile: Alfaomega Grupo Editor, S.A. - Av. Providencia 1443, Oficina 24, Santiago, Chile Tel.: (56-2)2235-4248 Fax: (56-2)2235-5786 - E-mail: agechile@alfaomega.cl

Argentina: Alfaomega Grupo Editor Argentino, S.A. Paraguay 1307 P.B. Of. 11, C.P. 1057, Buenos Aires, Argentina Tel./Fax: (54-11)4811-0887 y 4811 7183 - E-mail: ventas@alfaomegaeditor.com.ar

Acerca de los autores

Dr. Fernando Reyes Cortés: investigador titular C de la Facultad de Ciencias de la Electrónica en la Benemérita Universidad Autónoma de Puebla (BUAP). En 1984 obtiene el grado de Licenciado en Electrónica, en la Facultad de Ciencias Físico-Matemáticas de la BUAP; en 1990 realiza la Maestría en Ciencias con Especialidad en Electrónica, en el Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE) y el grado de Doctor en Ciencias en Electrónica y Telecomunicaciones lo obtiene en 1997 en el Centro de Investigación Científica y de Educación Superior de Ensenada (CICESE). Pertenece al Sistema Nacional de Investigadores desde 1993 a la fecha: nivel I. Desde 1980 se encuentra laborando en la BUAP. Es titular de los cursos de robótica de las carreras de



Ingeniería Mecatrónica y Licenciatura en Electrónica, así como en el Posgrado de Automatización de la Facultad de Ciencias de la Electrónica. Premio Estatal Puebla 2000 y Mérito Civil en Ciencias de la Ingeniería 2010 por el Ayuntamiento de la Ciudad de Puebla. Autor de múltiples artículos científicos nacionales e internacionales, ha titulado a más de 150 alumnos de los niveles de ingeniería/licenciatura, maestría y doctorado; ha diseñado más de 50 prototipos robóticos y mecatrónicos.



Dr. Jaime Cid Monjaraz: profesor investigador titular C del Centro Universitario de Vinculación y Transferencia Tecnológica (CUVyTT-DITCo) de la Benemérita Universidad Autónoma de Puebla (BUAP). Egresado de la Licenciatura en Electrónica en la Facultad de Ciencias Físico-Matemáticas (BUAP); Maestría en Control Automático por el Instituto Tecnológico de Puebla (ITP) y Doctorado en Ingeniería Mecatrónica en la Universidad Popular Autónoma del Estado de Puebla (UPAEP). Desde 1981 a la fecha es profesor en la BUAP de cursos de matemáticas, computación, electrónica, robótica y control en la Facultad de Ciencias de la Electrónica (BUAP). Autor de varias publicaciones científicas nacionales e internacionales. Ha desarrollado múltiples sistemas mecatrónicos y robóticos.

Fernando Reyes Cortés

Por todo el amor, apoyo y paciencia recibida, así como la comprensión de mi familia: para mi esposa Silvia y mis hijos Luis Fernando y Leonardo.

A mi Alma Mater, la Benemérita Universidad Autónoma de Puebla.

Jaime Cid Monjaraz

Mi más profundo agradecimiento y amor a mis hijas Grecia e Italia y a mi esposa Galia que son el principal motivo de mi felicidad. A toda mi familia cuya existencia es el acto más justo y hermoso. Por supuesto a mis colegas del CUVyTT-DITCo, a mis amigos y estudiantes de la BUAP que nada de este trabajo sería realidad sin su apoyo y de manera especial a la memoria del Ing. Luis Rivera Terrazas cuya influencia fue decisiva en mi vida.

Mensaje del Editor

Una de las convicciones fundamentales de Alfaomega es que los conocimientos son esenciales en el desempeño profesional, ya que sin ellos es imposible adquirir las habilidades para competir laboralmente. El avance de la ciencia y de la técnica hace necesario actualizar continuamente esos conocimientos, y de acuerdo con esto Alfaomega publica obras actualizadas, con alto rigor científico y técnico, y escritas por los especialistas del área respectiva más destacados.

Consciente del alto nivel competitivo que debe de adquirir el estudiante durante su formación profesional, Alfaomega aporta un fondo editorial que se destaca por sus lineamientos pedagógicos que coadyuvan a desarrollar las competencias requeridas en cada profesión específica.

De acuerdo con esta misión, con el fin de facilitar la comprensión y apropiación del contenido de esta obra, cada capítulo inicia con el planteamiento de los objetivos del mismo y con una introducción en la que se plantean los antecedentes y una descripción de la estructura lógica de los temas expuestos, asimismo a lo largo de la exposición se presentan ejemplos desarrollados con todo detalle y cada capítulo concluye con un resumen y una serie de ejercicios propuestos.

Además de la estructura pedagógica con que están diseñados nuestros libros, Alfaomega hace uso de los medios impresos tradicionales en combinación con las Tecnologías de la Información y las Comunicaciones (TIC) para facilitar el aprendizaje. Correspondiente a este concepto de edición, todas nuestras obras tienen su complemento en una página Web en donde el alumno y el profesor encontrarán lecturas complementarias así como programas desarrollados en relación con temas específicos de la obra.

Los libros de Alfaomega están diseñados para ser utilizados en los procesos de enseñanzaaprendizaje, y pueden ser usados como textos en diversos cursos o como apoyo para reforzar el desarrollo profesional, de esta forma Alfaomega espera contribuir así a la formación y al desarrollo de profesionales exitosos para beneficio de la sociedad, y espera ser su compañera profesional en este viaje de por vida por el mundo del conocimiento.

Contenido

| Plataforma de contenidos interactivos | XXI |
|--|------|
| | |
| Página Web del libro | XXII |
| | |
| Prólogo | XXXI |
| | |
| Capítulo 1 Introducción | 1 |
| 1.1 Introducción | Ę |
| 1.2 Sistemas empotrados | 4 |
| 1.3 Sistema empotrado Arduino | (|
| 1.3.1 Arquitectura abierta del sistema Arduino | (|
| шеь Ejemplos prácticos con Arduino | 13 |
| 1.4 Resumen | 14 |
| 1.5 Referencias selectas | 15 |
| 1.6 Problemas propuestos | 16 |
| Capítulo 2 Instalación y puesta a punto del sistema Arduino | 17 |
| | |
| 2.1 Introducción | 19 |
| 2.2 Instalación | 20 |
| 2.2.1 Instalación de drivers de las tarjetas Arduino | 21 |

| KII | | Contenido |
|----------------------|-----------------------------------|-----------|
| 2.3 Ambier | nte de programación Arduino | 24 |
| | Ienú Archivo | 26 |
| 2.4 Puesta | | 37 |
| | Ejemplo blink | 37 |
| | ejemplo DigitalReadSerial | 42 |
| | aplicaciones de blink | 44 |
| Web A | aplicaciones de DigitalReadSerial | 44 |
| Web A | aplicaciones del Sistema Arduino | 45 |
| 2.5 Resume | en | 46 |
| 2.6 Referen | icias selectas | 47 |
| 2.7 Probler | mas propuestos | 48 |
| | | |
| Capítulo Platafor | ma electrónica (leb | 49 |
| | | |
| 3.1 Introdu | acción | |
| 3.2 Arquite | ectura AVR | |
| 3.3 Platafo | rma electrónica Arduino | |
| 3.4 Modelo | s de tarjetas Arduino | |
| 3.5 Resume | en | |

| Capítulo 4 Lenguaje C | 51 |
|---|--|
| 4.1 Introducción | 53 |
| 4.2 Empezando a programar en C | 55 |
| 4.2.1 Operadores básicos del lenguaje C | 58 |
| Alfaomega Arduino. Aplicaciones en Robótica y Mecatrónica | Fernando Reyes Cortés • Jaime Cid Monjaraz |

3.6 Referencias selectas3.7 Problemas propuestos

| Contenido | XIII |
|---|------|
| 4.2.2 ¿Cómo ejecutar programas o sketchs? | 61 |
| | |
| 4.3 Variables | 61 |
| 4.3.1 Tipos de datos | 64 |
| 4.3.2 Modificadores de tipos de datos | 68 |
| 4.3.3 Constantes para cadenas y de la plataforma Arduino | 73 |
| 4.3.4 Ámbito de las variables | 74 |
| 4.4 Operadores | 76 |
| 4.4.1 Operadores aritméticos | 76 |
| Operadores a nivel de bits | 84 |
| Operadores lógicos y de comparación | 84 |
| 4.5 Arreglos | 85 |
| 4.5.1 Arreglos unidimensionales | 85 |
| 4.5.2 Arreglos bidimensionales | 86 |
| 4.6 Funciones | 87 |
| Sintaxis de funciones en lenguaje C | 88 |
| Ejemplos de funciones | 88 |
| 4.7 Instrucciones de programación | 94 |
| 4.7.1 Instrucciones condicionales | 94 |
| Ueb Instrucción if | 97 |
| Ejemplos con if | 97 |
| | 100 |
| Usb Instrucción if anidada | 100 |
| 4.7.2 Instrucción switch (valor){ case: break; default:} | 102 |
| Ejemplos con el operador ? | 103 |
| Ejemplos con switch(){} | 105 |
| 4.7.3 for(;;){} | 106 |
| Sintaxis de for $(;;)$ | 109 |

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

Fernando Reyes Cortés • Jaime Cid Monjaraz

| | Contenido |
|-----|-----------|
| XIV | Contenido |
| | |

| | 4.7.4 Instrucción while () $\{\}$ | 113 |
|------|---------------------------------------|-----|
| | 4.7.5 Sintaxis do $\{\}$ while(); | 115 |
| | 4.7.6 Sentencia break | 119 |
| | 4.7.7 Sentencia continue | 119 |
| | Ejemplos adicionales | 120 |
| 4.8 | Resumen | 121 |
| 4.9 | Referencias selectas | 121 |
| 4.10 | Problemas propuestos | 122 |
| | | |

Capítulo 5

Apuntadores, estructuras y uniones



125

- 5.1 Introducción
- 5.2 Apuntadores
- 5.3 Estructuras
- 5.4 Uniones
- 5.5 Resumen

Alfaomega

- 5.6 Referencias selectas
- 5.7 Problemas propuestos

Capítulo 6 Librerías y funciones Arduino

127

| 6.1 | Introducción | 129 |
|-----|-------------------------------|-----|
| 6.2 | Librerías Arduino | 130 |
| Mep | Librerías y funciones Arduino | 130 |
| | 6.2.1 Librerías stdio.h | 132 |
| | 6.2.2 Librerías stdlib.h | 132 |
| | 6.2.3 Funciones matemáticas | 135 |
| 6.3 | Funciones Arduino | 138 |
| | | |

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

Fernando Reyes Cortés • Jaime Cid Monjaraz

| Contenido | | XV |
|------------------|--|-----|
| | | |
| | Funciones fundamentales | 138 |
| | Utilidades | 139 |
| | Tipos de conversión | 139 |
| 6.3.4 | Funciones para puertos digitales entrada/salida | 141 |
| Шер | Manipulación de bits | 145 |
| 6.3.5 | Funciones para entradas analógicas | 149 |
| 6.3.6 | Características de los pins de entradas analógicas | 154 |
| Web | Adquisición de señales analógicas | 156 |
| Шер | Termómetro | 156 |
| 6.3.7 | Funciones time | 165 |
| 6.3.8 | Funciones matemáticas | 166 |
| Web | Aplicaciones de la función $\operatorname{map}(\dots)$ | 169 |
| 6.3.9 | Funciones para generar y detener tonos | 173 |
| 6.3.10 | Funciones para procesar bits y bytes | 178 |
| 6.3.11 | Serial | 179 |
| Web | Librerías estándar C | 187 |
| Mep | Librerías Arduino | 187 |
| Mep | Funciones Arduino | 187 |
| Mep | Interrupciones y aplicaciones | 187 |
| 6.4 Resu | men | 187 |
| 6.5 Refer | rencias selectas | 188 |
| 6.6 Prob | lemas propuestos | 189 |
| C4 | 1. 7 | |
| Capítu Servos | 10 / | 191 |
| | | |
| 7.1 Intro | | 193 |
| 7.2 Moto | ores de corriente directa | 194 |

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

Fernando Reyes Cortés • Jaime Cid Monjaraz

| Cont | enid |
|------|------|
| Con | t |

| | 7.2.1 Motor shield | 197 |
|-------|---|--|
| | 7.2.2 Librería Servo.h | 206 |
| | Librería Servo.h | 207 |
| 7.3 | Motores a pasos | 210 |
| | Motores a pasos | 212 |
| | 7.3.1 Párametros importantes de los motores a paso | os 213 |
| | 7.3.2 Motores a pasos con magneto permanente | 214 |
| | 7.3.3 Motores a pasos unipolares | 215 |
| | 7.3.4 Motores a pasos bipolares | 222 |
| | 7.3.5 Librería Stepper.h | 236 |
| | Ejemplos con motorreductores | 240 |
| | Ejemplos con motores a pasos | 240 |
| 7.4 | Resumen | 240 |
| 7.5 | Referencias selectas | 242 |
| 7.6 | Problemas propuestos | 243 |
| | pítulo 8 duino con Матьав | 245 |
| 8.1 | Introducción | 247 |
| | Información Arduino en MATLAB | 248 |
| | Integración numérica | 256 |
| | Diferenciación numérica | 262 |
| | Registro de resultados de trabajo | 266 |
| Web | Protocolo de comunicación | 269 |
| Web | Adquisición de datos MATLAB | 274 |
| Web | Arduino desde MATLAB | 274 |
| 8.6 | Resumen | 276 |
| FAOME | EGA ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA | Fernando Reyes Cortés • Jaime Cid Monjaraz |

https://dogramcode.com/libros-de-electronica

| Contenido | XVI |
|---|----------|
| | |
| 8.7 Referencias selectas | 27 |
| 8.8 Problemas propuestos | 27 |
| Capítulo 9 Control | 279 |
| 9.1 Introducción | 28 |
| 9.2 Sistemas de segundo orden | 28 |
| 9.2.1 Ecuación en variables de estado | 28 |
| Ejemplos de sistemas discretos | 28 |
| Simulación de sistemas dinámicos | 30 |
| Ejemplos con Arduino Due | 30 |
| 9.2.2 Aspectos técnicos a considerar en las tarjetas Arduino | 31 |
| Algoritmos de control | 31 |
| Control de un péndulo | 31 |
| 9.3 Control de temperatura | 31 |
| 9.3.1 Control de temperatura PID | 31 |
| 9.3.2 Regla de sintonía del control de temperatura PID | 31 |
| 9.3.3 Implementación práctica del control PID | 31 |
| Ejemplos con Intel Galileo | 32 |
| 9.4 Resumen | 33 |
| 9.5 Referencias selectas | 33 |
| 9.6 Problemas propuestos | 33 |
| Capítulo 10 Bluetooth | 333 |
| 10.1 Introducción | 33 |
| 10.1 Introduccion 10.2 Bluetooth | 33 |
| Annuno Anus granza ny Banárza v Mraumána. Empueno Prura Carrián - Leur Gr. Nacestra | 7 Armana |

XVIII Contenido

| 10.2.1 Arquitectura de los dispositivos Bluetooth | 338 |
|--|------------|
| 10.2.2 Especificaciones técnicas | 339 |
| 10.2.3 Aplicaciones Bluetooth | 339 |
| 10.3 Librerías para comunicación serial | 341 |
| 10.3.1 Librería SoftwareSerial del Sistema Arduino | 341 |
| 10.3.2 Módulo de Bluetooth JY-MCU | 344 |
| 10.3.3 Funciones de puerto serial Bluetooth de MATLAB | 353 |
| 10.4 Bluetooth Arduino+ MATLAB | 357 |
| Comunicación inalámbrica | 365 |
| Ejemplos ilustrativos | 372 |
| Web Aplicaciones de control | 372 |
| 10.5 Resumen | 372 |
| 10.6 Referencias selectas | 374 |
| 10.7 Problemas propuestos | 375 |
| Capítulo 11 Ethernet | 377 |
| 11.1 Introducción | 379 |
| 11.2 Tecnología de Ethernet | 380 |
| 11.3 Trama de Ethernet | 387 |
| 11.4 Arduino Ethernet Shield | 391 |
| 11.5 Librería Ethernet | 394 |
| 11.5.1 Ethernet: EthernetServer | 395 |
| 11.5.2 Ethernet: Client class | 397 |
| Configuración Cliente | 400 |
| Configuración Servidor | 400 |
| 11.6 Firmula and time | |
| 11.6 Ejemplos prácticos | 400 |
| Ejemplos Ethernet | 400 405 |

Alfaomega

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

FERNANDO REYES CORTÉS • JAIME CID MONJARAZ

| Contenido | XIX |
|---------------------------|-----|
| | |
| 11.7 Resumen | 415 |
| 11.8 Referencias selectas | 416 |
| 11.9 Problemas propuestos | 417 |
| | |

Capítulo 12 Manejo de interrupciones



419

421

- 12.1 Introducción
- 12.2 Tipos de interrupciones
- 12.3 Rutinas de servicio de interrupciones
- 12.4 Aplicaciones de control en tiempo real
- 12.5 Resumen
- 12.6 Referencias selectas
- 12.7 Problemas propuestos

Capítulo 13 WiFi

- 13.1 Introducción
- 13.2 WiFi
- 13.3 Puntos de acceso
- 13.4 WiFi Shield
- 13.5 Resumen
- 13.6 Referencias selectas
- 13.7 Problemas propuestos



- 14.1 Introducción
- 14.2 Ambiente de programación LabVIEW

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

Fernando Reyes Cortés • Jaime Cid Monjaraz

XX Contenido

- 14.3 Programación LabVIEW
- 14.4 Adquisición y desplegado de datos
- 14.5 Resumen
- 14.6 Referencias selectas
- 14.7 Problemas propuestos

Índice analítico

425

Alfaomega

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

FERNANDO REYES CORTÉS • JAIME CID MONJARAZ

Plataforma de contenidos interactivos

Para tener acceso al material de la plataforma de contenidos interactivos de **Arduino. Aplicaciones en Robótica y Mecatrónica**, siga los siguientes pasos:



1) Ir a la página

http://libroweb.alfaomega.com.mx



2) Registrarse como usuario del sitio.



3) Ingresar al apartado de inscripción de libros, o bien identificar este libro en el catálogo, y registrar la siguiente clave de acceso.



4) Para navegar en la plataforma ingrese los nombres de Usuario y $Contrase\~na$ definidos en el punto dos.

Página Web del libro

La página Web de la obra **Arduino. Aplicaciones en Robótica y Mecatrónica** contiene los siguientes recursos:



Videos experimentales

Videos experimentales de sistemas mecatrónicos y prototipos científicos que se encuentran disponibles en el sitio Web del libro para mostrar aspectos cualitativos de diversos conceptos académicos.



Videos de prototipos científicos desarrollados en el laboratorio de Robótica de la Facultad de Ciencias de la Electrónica de la *Benemérita Universidad Autónoma de Puebla* y que en general son la implementación de la teoría de automatización aplicada a la ingeniería robótica y mecatrónica.



Simuladores

Como un paso previo a la etapa experimental, se encuentra el estudio y análisis de sistemas dinámicos y algoritmos de control a través de simuladores de robots y servomecanismos de ingeniería mecatrónica. Se proporciona el código fuente del modelo dinámico del sistema, así como numerosos ejemplos de aplicación.



Código fuente

Se incluyen un número importante de programas desarrollados en código fuente **MATLAB** versión 2014a y debidamente documentados para aplicarse en: instrumentación y acondicionamientos de señales, sistemas dinámicos lineales y no lineales, robots manipuladores, servomecanismos, algoritmos de control, diagramas fase, control clásico, sistemas discretos, etc.



Lecturas complementarias

Adicional a los 14 capítulos que integran la obra, también se incluyen en el sitio Web del libro diversos documentos con temas relacionados con la ingeniería robótica y mecatrónica, así como sus aplicaciones.



Respuesta y desarrollo de problemas seleccionados

Para el estudiante se encuentra en la página Web la solución de una selección de ejercicios planteados en los capítulos del libro. En la solución de dichos ejercicios se detallada los pasos teóricos y su implicaciones prácticas.

Recursos Web y notas al margen

Dentro de la composición de la presente obra, se encuentran una serie de notas al margen con la finalidad de explicar conceptos claves, evitando que el lector se distraiga de la lectura para investigar o consultar en otro lado ese concepto. Asimismo, también se dispone de los recursos Web, cuyo material se encuentra en formato electrónico dentro del sitio Web de la Editorial Alfaomega.



Notas al margen

Notas al margen son recuadros en tonos de gris, con un icono en forma de tachuela insertada en la esquina superior izquierda del recuadro. Notas al margen, tienen la finalidad de describir aspectos claves e importantes, que complementan la sección en desarrollo y cuyos conceptos se describen como apoyo al lector.

Los recursos Web se distinguen por un recuadro en tonos de gris y un icono indicando que hace referencia a que se encuentra material disponible en el sitio Web destinado a esta obra.

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

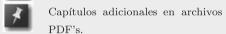
Fernando Reyes Cortés • Jaime Cid Monjaraz



Recursos Web

Los recursos Web que incluye esta obra





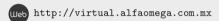
Lecturas adicionales en archivos PDF's.

Videos experimentales o descripti-

Hojas de especificaciones.

Solución de problemas propuestos seleccionados.

Para acceder a los recursos Web del libro:



Identificar la obra del catálogo, bajar la información complementaria y adicional del capítulo.



Simbología e iconografía utilizada

Con el fin de que el lector identifique fácilmente la sintaxis y descripción de instrucciones o comandos de programación de Arduino y MATLAB que se presentan en el libro se utilizan las siguientes formas de simbología:

Arduino se identifica con el símbolo matemático de infinito: .

FERNANDO REYES CORTÉS • JAIME CID MONJARAZ

Alfaomega

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

Por lo tanto, para describir funciones específicas de Arduino se emplea el siguiente cuadro:

char(dato)



Para instrucciones y funciones de MATLAB, empleamos lo siguiente:

fplot('function',limits,line)



Ejemplos

Los ejemplos ilustrativos que se encuentran resueltos en la obra están clasificados con respecto a su grado de complejidad en tres formas posibles: simple, regular y complejo. La solución de todos los ejemplos ha sido documentada cuidadosamente, para que el lector siga con detalle cada paso en el planteamiento de la respuesta.

& Ejemplo 1.1

Los ejemplos simples se encuentran identificados por un símbolo \$\,\$, complejidad regular por \$\,\$\$ y complejos por \$\,\$\$. También incluyen un número de referencia que lo identifica al capítulo donde fue definido. El enunciado de los ejemplos se encuentra en un recuadro con fondo gris; también se indica el recuadro de solución y la terminación del ejemplo por 3 cuadros posicionados a la derecha del margen.

Solución

Se describe la solución a detalle del planteamiento del problema, cuidado en cada paso justificar académicamente lo que sustenta dicha propuesta, la cual incluye formulación, descripción de conceptos, programas y gráficas de resultados. Se finaliza el desarrollo del mismo con una serie de tres cuadros pequeños posicionados a la derecha de la página, los cuales definen la separación con los siguientes párrafos.



Programas

Todos los programas de este libro han sido implementados en lenguaje C para el sistema Arduino, indicando en las correspondientes secciones el tipo de versión del paquete Arduino; para MATLAB el desarrollo del código fuente es utilizando la versión 2014a. La presentación y documentación de los programas se encuentran identificados por cuadros de código en tonos de gris para Arduino

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

Fernando Reyes Cortés • Jaime Cid Monjaraz

y MATLAB con número de referencia, cabecera con información del capítulo al que pertenecen, nombre del sketch (Arduino) y script (MATLAB) y sus respectivas notas de documentación. Por ejemplo, para describir sketchs se emplea el siguiente cuadro de código Arduino :

```
Código Arduino 6.1: sketch cap6_digitalports
  Arduino. Aplicaciones en Robótica y Mecatrónica.
                                                        Disponible en Web
  Capítulo 6 Librerías y funciones Arduino.
  Fernando Reyes Cortés y Jaime Cid Monjaraz.
  Alfaomega Grupo Editor: "Te acerca al conocimiento".
   Sketch cap6_digitalports.ino
 1 int pin5=5, pin6=6, pin7=7, pin11=11, pin12=12, pin13=13;//asignación de puertos.
 2 int i, j;//variable i es el pivote de la instrucción for(; ;){...}, j registra resultado.
 3 void setup() {//subrutina de configuración.
       Serial.begin(9600); //velocidad de transmisión 9600 Baudios.
       pinMode(pin5, INPUT); //pin 5 configurado como entrada digital.
       pinMode(pin6, INPUT); //pin 6 configurado como entrada digital.
 6
 7 }
 8 void loop() {//lazo principal del sketch.
       for(i=0; i<7; i++){}
 9
          digitalWrite(pin13, 0x0001&i);//pin 13 con pin 5.
10
          digitalWrite(pin12, 0x0002&i);//pin 12 con pin 6.
11
          digitalWrite(pin11, 0x0004&i);//pin 11 con pin 7.
12
         j=0x0001&digitalRead(pin5);//lectura del pin 5 con máscara del primer bit.
13
         j=j|digitalRead(pin6)<<1;//lectura del pin 6 y corrimiento a la izquierda.
14
          //Los 16 bits de j tienen la forma: j=0000 0000 0000 0 pin7 pin6 pin5.
15
         j=j| digitalRead(pin7)<<2;//lectura del pin 7 y corrimiento a la izquierda.
16
          Serial.println("Salida,
                                   Entrada"); Serial.print(i); delay(10);
17
          Serial.print("\t \t \"); delay(10);
18
          Serial.println(j); delay(10);
19
       }// fin de la instrucción for (;;) \{...\}.
20
       Serial.println("En 5 segundos, se ejecutará una vez más el sketch");
21
       delay(5000);
22
23 }
```

Observe que el código fuente se encuentra numerado por línea, esto facilita la descripción y la explicación de los sketchs.

ALFAOMEGA ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA FERNANDO REYES CORTÉS • JAIME CID MONJARAZ

De manera similar para cuadros de código MATLAB :



```
A Código MATLAB 8.4 cap8_graficaInt.m
  Arduino. Aplicaciones en Robótica y Mecatrónica.
  Capítulo 8 Comunicación con MATLAB.
  Fernando Reyes Cortés y Jaime Cid Monjaraz.
  Alfaomega Grupo Editor: "Te acerca al conocimiento".
   Archivo cap8_graficaInt.m
                                                           Versión de Matlab 2014a
 1 clear all close all clc
 2 format short
 3 canal-serie = serial('COM3', 'BaudRate', 9600, 'Terminator', 'CR/LF'); %crear objeto serie.
 4 fopen(canal_serie); %abrir puerto.
 5 xlabel('Segundos');
 6 ylabel('Datos');
 7 title('Adquisición de datos Arduino');
 8 grid on; hold on;
 9 prop = line(nan,nan, 'Color', 'b', 'LineWidth',1);
10 datos = fscanf(canal_serie, '%f %f %f ,[3,1]); %leer el puerto serie.
12 disp('Adquisición de datos de la tarjeta Arduino UNO');
13 i=1;
14 while i<10000
15
       datos = fscanf(canal_serie, '%f %f %f ,[3,1]); %leer el puerto serie.
       Int_k(i) = datos(1,1);
16
       Int(i) = datos(2,1);
17
       tiempo(i) = datos(3,1);
18
       set(prop, 'YData', Int(1:i), 'XData', tiempo(1:i));
       drawnow;
20
      i=i+1;
\mathbf{21}
22 end
23 figure
24 plot(tiempo,Int, tiempo, Int_k)
```

Arduino. Aplicaciones en Robótica y Mecatrónica

27 clear canal_serie;

25 fclose(canal_serie); %cierra objeto serial. 26 delete(canal_serie); %libera memoria.

Fernando Reyes Cortés • Jaime Cid Monjaraz

También hay recuadros de código Arduino que son ejemplos sencillos para ilustrar ideas y conceptos claves del lenguaje C, como el que a continuación se describe:

∞

Código ejemplo 4.2

Variables globales y locales

```
int x,y,z; //variables globales, visibles por cualquier función y en cualquier parte del programa.
void setup(){
    int i,j,k; //variables locales a la función setup()
    i=0;
    j=9;
    k=i*j;
}
void loop(){
    float f; // la variable f sólo es visible dentro de loop().
    f=f+10;
    x=f*10;
    y=f/100; z=f-100;
}
```

Cajas de texto

Las cajas de texto son recuadros en tonos de gris con la finalidad de remarcar avisos, notificación, métodos de instalación y procedimientos, y en tal caso sugerencias para el correcto desarrollo de un programa o algoritmo. En otras palabras, se pretende llamar la atención del lector para indicarle un procedimiento o método que le facilite implementar un determinado algoritmo.

Ejemplo de una caja de texto es la siguiente representación:

Caja de texto



Describe procedimientos para configurar tarjetas electrónicas.



Contiene métodos numéricos para obtener observar señales de velocidad.



Instalación de paquetes de cómputo.



Indica los pasos secuenciales para editar, compilar, descargar y ejecutar un sketch en una tarjeta electrónica Arduino.



También puede tener la definición de un concepto académico o de una norma de tecnología.

Alfaomega

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

FERNANDO REYES CORTÉS • JAIME CID MONJARAZ

Referencias bibliográficas

Los símbolos empleados en las referencias tales como libros, revistas especializadas o técnicas y enlaces electrónicos como sitios Web son representados mediante los siguientes:



Se emplea para identificar un libro o proceedings.



Utilizado para identificar una revista científica ya sea Journal o Transactions.



Indica una dirección electrónica.

Prólogo

A que utilizan microcontroladores para formar sistemas empotrados especializados en la automatización de procesos, con un amplio espectro de aplicaciones en ciencias exactas y en ingeniería. Este tipo de plataformas se han popularizado en todo el mundo, gracias al desempeño, costo y arquitectura abierta de sus modelos; esto último, es la característica principal que lo hace atractivo para desarrollar aplicaciones de automatización, ya que admite adaptabilidad para automatizar cualquier proceso físico. Otra de las ventajas del sistema Arduino, es que tiene software libre (gratuito), así como el acceso y adecuación del código fuente y de la electrónica de sus tarjetas.

Actualmente, Arduino se ha posicionado como apoyo didáctico y pedagógico de los cursos de física, matemáticas, robótica, mecatrónica, control, instrumentación, sistemas dinámicos y en general para la mayoría de las áreas de la ingeniería representa una herramienta importante y fundamental. Hoy en día, vemos que los catedráticos de la mayoría de la Universidades del mundo presentan diversos proyectos para que sus alumnos lo realicen con Arduino. El potencial que tiene los lenguajes C/C++, microcontroladores de arquitectura AVR, periféricos como convertidores A/D, comunicación USB, timers y puertos digitales I/O, interrupciones, módulos para controlar servomotores y motores a pasos, comunicación inalámbrica (Bluetooth y WiFi), y por Ethernet, desarrollo de un número importante de librerías, así como el soporte técnico requerido para llevar a cabo aplicaciones complicadas, hacen del sistema Arduino una opción atractiva para utilizarlo en varios aspectos de la vida cotidiana.

Arduino nació en el año 2005 como una necesidad de proponer un sistema electrónico para instrumentación y automatización barato y con arquitectura abierta; el equipo fundador ha sido: Massimo Banzi, David Cuartilles, Gianluca Martino, Tom Igoe y David Mellis. Arduino tiene varios tipos de tarjetas electrónicas, iniciando con la básica, el modelo UNO, posteriormente llegaron los modelos Leonardo, Due, Mega 2560, Yún, Tre, Zero, LilyPad, y en el año 2014, nació el modelo Galileo, el cual contiene el microcontrolador Intel Quark Soc X1000 Application Processor de 32 bits Pentium y con 400 MHz de velocidad de trabajo.

El sistema Arduino tiene un ambiente de programación (IDE) con herramientas integradas para editar, compilar y descargar sketchs a las tarjetas electrónicas. Cuenta también con suficiente información y documentación técnica, así como ejemplos de desarrollo. Además, debido a su lenguaje de programación (C/C++) y las características tecnológicas de la plataforma electrónica es muy fácil expandirlo a otros paquetes de programación como **MATLAB** y LabVIEW.

Con todas las características ya citas sobre el sistema Arduino, constituye una poderosa herramienta para impactar en las materias que conforman el plan de estudios de las carreras de ingeniería

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

FERNANDO REYES CORTÉS • JAIME CID MONJARAZ

XXXII Prólogo

y ciencias exactas; aún más, también tiene el potencial para emplearse en el sector industrial, comercial y doméstico. En la literatura científica, ya es común ver citas y referencias sobre el desarrollo tecnológico de un prototipo científico con Arduino; esto involucra a los posgrados dentro de los niveles de maestría y doctorado, lo cual representa un indicativo global.

La presente obra está enfocada a describir el sistema Arduino para cubrir tópicos y temas de la currícula del plan de estudios de las carreras de ingeniería: mecatrónica, robótica, electrónica, eléctrica, automatización, informática, industrial, computación y sistemas. Este libro desarrolla en forma integral las áreas de la automatización aplicadas a la ingeniería robótica y mecatrónica sin perder claridad y calidad académica en el tratamiento y exposición de los temas; al mismo tiempo enriquece la parte práctica con propuestas teóricas-experimentales. Por tal motivo, la presente obra cubre en forma horizontal los niveles básicos e intermedios del plan de estudios y en forma transversal es una excelente referencia que sirve al alumno para toda su carrera y vida profesional.

Dentro de las finalidades de este libro, se encuentra brindar al lector las bases fundamentales de la ingeniería robótica y mecatrónica abordando aspectos de programación en los lenguajes C y MATLAB, control clásico y control moderno bajo el enfoque de espacio de estados, así como modelado de sistemas continuos, discretos y control de sistemas mecatrónicos. Se presentan un número importante de ejercicios resueltos en forma analítica y bien documentados para ayudar al lector a comprender y mejorar los conocimientos presentados en las respectivas secciones; además, por medio de simulación como herramienta didáctica refuerce los conocimientos y le permita verificar los resultados prácticos que predice la teoría.

El contenido de este libro es el resultado del proceso de aprendizaje, desarrollo, generación y aplicación del conocimiento que han tenido los autores, quienes se caracterizan por tener más de 30 años de experiencia como docentes, algo también relevante de comentar es que son científicos, con dominio completo sobre teoría y tecnología en robótica, mecatrónica y control automático; ingredientes que combinan en forma adecuada la técnica y estilo para exponer, presentar y transmitir los temas seleccionados.



Organización del libro

Este libro se encuentra organizado en 14 capítulos fundamentales del área de la robótica y mecatrónica que de manera integral cubren los aspectos teóricos y prácticos de los planes de estudios relacionados con las carreras de ingeniería. Está dividido en tres partes, Parte I: sistema Arduino (capítulos 1 al 3); Parte II: lenguaje de programación (capítulos 4 al 6) y Parte III: aplicaciones (capítulos 7 al 14).

Alfaomega

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

FERNANDO REYES CORTÉS • JAIME CID MONJARAZ

Prólogo XXXIII

En forma impresa se describen 9 capítulos; adicionalmente, la obra plantea cinco capítulos Web que por razones de espacio en el material impreso, fueron destinados al formato electrónico en la página Web de este libro, enriquecidos con una variedad de material adicional y recursos Web.

A continuación se describe en forma sucinta el contenido de los 14 capítulos que componen a esta obra.



Parte I: sistema Arduino

La primera parte del libro está compuesta por 3 capítulos que describen la filosofía de trabajo e impacto, ambiente de programación (instalación y puesta a punto) y características de la plataforma electrónica del Sistema Arduino.

Capítulo 1 Introducción

Presenta una reseña histórica, importancia, enlaces para acceder a la página web de 🐼 y describe la filosofía de trabajo del sistema Arduino; resaltando sus características claves de arquitectura abierta y software libre. Desarrolla los aspectos estratégicos de los sistemas empotrados, detallando los conceptos fundamentales de este tema.

Capítulo 2 Instalación y puesta a punto del sistema Arduino

Se describe paso a paso el proceso de instalación del entorno de programación Arduino (IDE), así como los drivers para reconocer la comunicación USB entre la computadora y la tarjeta Arduino. Asimismo, se desarrolla el manual de usuario del ambiente de programación, explicando cada una de sus herramientas de trabajo, aspectos cualitativos del comando cargador de código que permite la transferencia del sketch hacia la tarjeta Arduino. La puesta a punto, significa que la computadora que tiene instalado el IDE reconozca correctamente a la tarjeta Arduino y que funcionen completamente todos los comandos del ambiente de programación tales como: edición, compilación, descarga y ejecución del sketch en las tarjetas Arduino, así como la visualización de resultados en la ventana monitor. Este proceso se verifica con la comprobación de dos ejemplos prácticos de Arduino: blink y DigitalReadSerial.

Capítulo 3 Plataforma electrónica

Este es un capítulo de que describe la arquitectura AVR de los microcontroladores ATMEL; se presenta la tabla y manejo de interrupciones, organización de la memoria de las tarjetas, tipos de periféricos disponibles en las tarjetas electrónicas, formas de comunicación serial, señales para

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

FERNANDO REYES CORTÉS • JAIME CID MONJARAZ

XXXIV Prólogo

interface y la descripción de los modelos de tarjetas Arduino.



Parte II: lenguaje de programación

La segunda parte de esta obra comprende la programación en lenguaje C (gramática y sintaxis) y la descripción de librerías y funciones Arduino. A diferencia con otras obras sobre lenguaje C, aquí presentamos la exposición del lenguaje C específico para Arduino, de tal forma que el lector podrá comprobar que todo lo que ha aprendido funciona realmente en las tarjetas electrónicas y por lo tanto el grado de entendimiento y madurez en este lenguaje será mayor.

Capítulo 4 Lenguaje C

La exposición de conceptos del lenguaje C se realiza a través de bloques y diagramas de flujo para explicar los aspectos cualitativos, los resultados y detalles de manipulación de datos, por medio de operadores, instrucciones y funciones se verifican con el apoyo de una variedad de ejercicios bien documentados; en esta fase, el lector empieza con la edición del sketch en el ambiente integrado Arduino (IDE), compilación y descarga del código de máquina hacia la tarjeta electrónica. Los resultados de la ejecución del sketch se visualizan dentro del entorno de programación, de esta forma, el lector se familiariza no sólo con las tarjetas Arduino, también asimila la gramática y sintaxis del lenguaje C, con dominio pleno para diseñar código eficiente que le permita automatizar procesos. Este capítulo describe los tipos de datos, operadores aritméticos y lógicos a nivel bytes y bits, arreglos, funciones, sentencias, instrucciones y estilo de programación.

Capítulo 5 Apuntadores, estructuras y uniones

Este es un capítulo web que describe los conceptos y aspectos técnicos para implementar aplicaciones con apuntadores, estructuras de datos y uniones. Se presentan varios ejemplos ilustrativos sobre la forma de programar variables puntero, inicialización y asignación dinámica de memoria, aritmética de apuntadores, estructura y uniones de datos, así como manipulación de sus campos.

Capítulo 6 Librerías y funciones Arduino

Se describen las funciones de las librerías Arduino, las cuales permiten aprovechar la enorme cantidad de recursos de la plataforma electrónica. El desempeño de Arduino se incrementa cuando se combinan las librerías Arduino con el poder de programación de los lenguajes C/C++ (tipos de datos, clases, operadores, instrucciones, funciones, librerías estándar), lo que resulta de un lenguaje de programación enriquecido para implementar cualquier aplicación en ciencias exactas, robótica y mecatrónica.

Alfaomega Ardu

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

FERNANDO REYES CORTÉS • JAIME CID MONJARAZ

Prólogo XXXV



Parte III: aplicaciones

La última parte de esta obra se ubica en el desarrollo de aplicaciones del sistema Arduino en ciencias exactas e ingenierías, abordando temas como: servos, comunicación y enlace con **MATLAB**, algoritmos de control, comunicación Bluetooth, Ethernet, manejo de interrupciones, WiFi y LabVIEW.

Capítulo 7 Servos

Este capítulo está destinado a presentar los fundamentos prácticos de una clase particular de actuadores eléctricos, denominados servos (motores de corriente directa y motores a pasos). Se describen las librerías Arduino para control de servos y también se desarrolla código propio que permite implementar aplicaciones con este tipo de motores a través de las tarjetas Arduino. Se presentan varios ejemplos didácticos.

Capítulo 8 Arduino con MATLAB

En el presente capítulo se desarrollan herramientas y protocolos de sincronía y enlace para comunicación USB entre las tarjetas Arduino con el ambiente de programación de MATLAB. Mientras que, en la plataforma electrónica de la tarjeta Arduino se encuentra en ejecución el sketch, en MATLAB se visualiza la información de interés al usuario en forma de texto o mediante su representación gráfica; así como, registrar en un archivo de datos esa información para propósitos de análisis, procesamiento o estudio detallado del sistema a controlar. Se propone un formato tabular para registrar la información de variables, sensores y señales en un archivo de datos.

Capítulo 9 Control

Se presentan técnicas para implementar algoritmos de control, sistemas discretos, métodos numéricos de integración y diferenciación e instrumentación electrónica básica para acoplamiento de sensores con señales de baja magnitud. Asimismo, se describe el comportamiento cualitativo de un sistema dinámico de segundo orden y el algoritmo proporcional integral derivativo (PID) para controlar un horno eléctrico. Se describen resultados de simulación y experimentales con tarjetas Arduino.

Capítulo 10 Bluetooth

Bluetooth es una técnica de comunicación inalámbrica con amplias aplicaciones en ingeniería. Se describen ejemplos ilustrativos que permiten comunicar inalámbricamente dispositivos remotos que

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

Fernando Reyes Cortés • Jaime Cid Monjaraz

XXXVI Prólogo

tienen el sistema Android (teléfonos celulares y tablets) y computadores con el Sistema Operativo Windows. Se implementan aplicaciones con sistemas discretos y control digital con librerías de comunicación Bluetooth entre Arduino y MATLAB.

Capítulo 11 Ethernet

Presenta la programación que permite la configuración de la tarjeta Ethernet Shield al protocolo de comunicación Web para expandir las aplicaciones que tiene el sistema Arduino en el ámbito global. Se describen ejemplos prácticos que ilustran la forma de desplegar información en una página Web. Asimismo, también se incluyen los siguientes puntos: programación de aplicaciones Web usando las funciones de la librería Ethernet; descripción de la tecnología Ethernet; asignación de direcciones IP para desplegar información de variables y procesos de automatización; características de programación para cliente/servidor; así como, visualización en una página Web de: adquisición de datos, cálculos numéricos y procesamiento de información.

Capítulo 12 Manejo de interrupciones

Capítulo de destinado a presentar el manejo de interrupciones en la plataforma electrónica de las tarjetas Arduino. Se describen el tipo de interrupciones que existen y ejemplos ilustrativos con subrutinas de servicio de interrupciones para aplicaciones en adquisición de datos y control de servomecanismos en tiempo real.

Capítulo 13 WiFi

Capítulo (describe la técnica de comunicación inalámbrica mediante WiFi, puntos de acceso, protocolo de comunicación y ejemplos didácticos con envío/transmisión de comandos vía WiFi para control de servomecanismos.

Capítulo 14 LabVIEW

Capítulo (Lab), el cual presenta la comunicación del sistema Arduino con el paquete LabVIEW. Se realizan aplicaciones para desplegar información de sensores de temperatura, infrarrojos, de proximidad, humedad y adquisición de señales analógicas.



Créditos de programas de cómputo utilizados

En la presente obra se ha tomado en cuenta los aspectos de calidad científica y académica docente, así como los aspectos pedagógicos de la exposición de conceptos, secuencia y desarrollo del contenido,

Alfaomega

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

FERNANDO REYES CORTÉS • JAIME CID MONJARAZ

Prólogo XXXVII

solución de ejemplos ilustrando y fortaleciendo los conocimientos, programación y desarrollo de simuladores. Por lo que, en la edición, formación y compilación del manuscrito fueron seleccionados un conjunto de herramientas y paquetes de cómputo cuya calidad estética es insuperable.

Créditos de programas y herramientas de cómputo utilizados

En este libro se ha priorizado la calidad de presentación no sólo en la exposición de los conceptos, también la estética y estilo de objetos pedagógicos, gráficos e iconografía con la finalidad de captar y motivar la atención de alumnos y profesores. Por tal motivo, la presente obra fue formada y editada en lenguaje científico LAT_EX y compilada con macros y programación desarrollada por los autores usando PCT_EX^{MR} 6.0; los diagramas y dibujos realizados en AutoCAD^{MR} 2014, imágenes de fritzing^{MR} 8.5 y sólidos diseñados en SolidWork^{MR} 2013, los programas fuentes de simuladores para sistemas dinámicos, mecatrónicos y algoritmos de control fueron realizados para MATLAB^{MR} versión 2014a y los programas en lenguaje C con el ambiente de programación integrado de Arduino^{MR} en las versiones: 1.0.5, 1.5.5, 1.5.7, 15.8 e Intel Galileo (1.5.3).

Palabras finales

Los autores desean agradecer a la Benemérita Universidad Autónoma de Puebla por todo el apoyo proporcionado para realizar esta obra, particularmente al Mtro. Alfonso Esparza Ortiz, rector de la Institución por su visión científica y liderazgo social, así como al Dr. Pedro Hugo Hernández Tejeda, titular de la Dirección de Innovación y Transferencia del Conocimiento (DITCo); al Dr. José Ramón Eguibar Cuenca, Director General de Investigación, de la Vicerrectoría de Investigación y Estudios de Posgrados (VIEP). Este libro, también fue posible gracias al apoyo de los proyectos: "Modelado Dinámico y Simulación de Robots Manipuladores" y "Control Visual de Robots" pertenecientes al sub-programa de aseguramiento de investigadores consolidados de la VIEP; asimismo, al programa de Innovación Tecnológica "Plataforma Robótica" impulsado por DITCo. Por útlimo, deseamos extender nuestro agradecimiento a todas aquellas personas que impulsaron y fortalecieron significativamente los conocimientos presentados en este libro.

Fernando Reyes Cortés
Facultad de Ciencias de la Electrónica

Jaime Cid Monjaraz
CUVyTT-DITCo

Puebla, Pue., a 27 de noviembre del 2014

Benemérita Universidad Autónoma de Puebla

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

FERNANDO REYES CORTÉS • JAIME CID MONJARAZ



Introducción



- 1.1 Introducción
- 1.2 Sistemas empotrados
- 1.3 Sistema empotrado Arduino
- 1.4 Resumen
- 1.5 Referencias selectas
- 1.6 Problemas propuestos

Competencias

Presentar la importancia y el impacto que ha tenido el sistema de programación Arduino orientado a la automatización de procesos físicos, así como sus aplicaciones en ciencias exactas e ingenierías.

Desarrollar habilidades en:

Descripción y características del sistema Arduino.

Plataforma electrónica.

Sistemas empotrados.

Filosofía del concepto de arquitectura abierta.

1.1 Introducción 3

1.1 Introducción



A RDUINO es una plataforma electrónica y de programación en arquitectura abierta con amplia gama de aplicaciones en ciencias exactas e ingeniería. El icono que identifica al sistema Arduino es el símbolo infinito, que se emplea en el área de matemáticas, insertando en su interior los signos - y +, es decir: La plataforma de este sistema Arduino gira alrededor de la familia de microcontroladores ATMEL y por medio de un entorno de programación, con las herramientas necesarias integradas para diseñar y desarrollar aplicaciones de manera sencilla en los lenguajes C y C++, representa un sistema empotrado (embedded system) de propósito específico para automatizar procesos físicos.

Hoy en día, Arduino se ha posicionado de manera vertiginosa como una poderosa herramienta tecnológica en la industria, universidades y centros de investigación, debido a la filosofía de arquitectura abierta, que lo hace un sistema perse para realizar automatización de procesos físicos "a la medida". Arduino pude ser usado para monitorear una enorme variedad de sensores analógicos y digitales, así como para controlar servomotores de corriente directa y alterna, a pasos, además de llevar a cabo aplicaciones con robots móviles e industriales, comunicación Wi-Fi, Bluetooh y Ethernet; debido a que cuenta con un puerto USB que expande sus prestaciones y potencialidades al comunicarse con otros paquetes como MATLAB, LabVIEW, entre otros.

El software del sistema Arduino es gratuito y se puede descargar del sitio **www.arduino.cc**, las tarjetas pueden ser ensambladas a mano o adquiridas como sistemas de desarrollo a precio muy económico.

Arduino toma ventaja de las prestaciones tecnológicas de la familia de microcontroladores ATMEL, desarrollando un entorno de programación en arquitectura abierta con la suficiente flexibilidad para la implementación de aplicaciones en ingeniería robótica y mecatrónica. El lenguaje de programación de Arduino se basa en C y C++, herramientas ideales para modelar, analizar, procesar y llevar a la práctica cualquier problema científico-tecnológico, al mismo tiempo, proporcionan al sistema Arduino sencillez y poder de cómputo, muy superior a otras plataformas similares que emplean otro tipo de lenguaje y soporte electrónico.

Como una consecuencia directa de las anteriores prestaciones, Arduino es un sistema que se caracteriza por tener las herramientas necesarias para simplificar el proceso de implementación práctica, permite al usuario aprovechar el tiempo para enfocarse específicamente al desarrollo de la automatización del proceso físico y no distraer la atención en aspectos técnicos de programación, como sucede con otros sistemas. Arduino ofrece a investigadores, profesores, ingenieros y estudiantes un soporte de desarrollo tecnológico con enorme diversidad de aplicaciones.

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

Fernando Reyes Cortés • Jaime Cid Monjaraz

4 Introducción

Particularmente en el área de docencia, Arduino se ha convertido en una herramienta pedagógica muy completa que facilita el proceso de transmisión de conocimientos en los cursos de microcontroladores, instrumentación electrónica, control clásico, servomecanismos, sistemas discretos, control digital, sistemas dinámicos lineales y no lineales, robótica móvil, robots manipuladores, física, circuitos eléctricos, etc.

La plataforma Arduino se distingue por las siguientes características técnicas:



Mantiene compatibilidad con los sistemas operativos: Windows, Macintosh OSX y Linux.



El software Arduino tiene licencia libre, es decir gratuita, descargándose directamente de:

www.arduino.cc



Arduino puede ser mejorado y ampliarse a través de la incorporación de librerías en los lenguajes C y C++; para profundizar en los detalles técnicos del sistema, es posible dar el salto a la programación en lenguaje AVR, en el que está basado y combinarlo con C y C++.



La plataforma electrónica se basa en la familia de microcontroladores ATMEL de 8 bits (ATmega48PA/88PA/168PA/328P) en arquitectura AVR-RISC, (Reduced Instruction Set Computing) cuya característica principal es lograr alto desempeño del CPU en la ejecución de programas debido a la simplicidad de programación por ciclo de reloj, aproximadamente ejecuta un millón de instrucciones por segundo (MIPS) por cada unidad de MHz; por otro lado, emplea memoria flash no volátil de alta densidad para almacenar programas con capacidad de lectura mientras escribe (Read-While-Write). Además, tiene 23 líneas digitales que se pueden programar como entrada/salida, así como 32 registros de trabajo de propósito general, 3 contadores (dos de 8 bits y uno de 16 bits), contador para tiempo real, interrupciones internas y externas, dispositivo de comunicación serie USART y un convertidor analógico/digital de 10 bits conectado a un multiplexor de 8 canales.



Los diagramas esquemáticos y módulos de componentes están publicados bajo licencia de *Creative Commons*, por lo que diseñadores de sistemas electrónicos pueden realizar su propia versión del módulo, ampliándolo u optimizándolo.



1.2 Sistemas empotrados

Os sistemas empotrados (*embedded systems*) son plataformas electrónicas, cuya base computacional está constituida por un microcontrolador con el lenguaje de programación adecuado y un conjunto de periféricos (puertos I/O, timers, contadores, PWM's, FPGA's) e instrumentación electrónica (convertidores analógico/digital y digital/analógico) para acoplamiento de señales y comandos de sensores, servomotores, robots, etcétera; este sistema está dedicado exclusivamente al procesamiento, control y automatización de procesos físicos en tiempo real.

Alfaomega

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

Control de procesos en tiempo real

El control de procesos físicos en tiempo real se refiere al tiempo de máquina que emplea el microcontrolador para realizar todas las operaciones aritméticas, adquisición de datos, procesamiento de la información y envío de comandos, de tal forma, que este tiempo de cómputo debe ser menor al periodo de muestreo.

La respuesta de un sistema en tiempo real depende de realizarlo en el menor tiempo posible, en el caso de que existan retardos, éstos deberán ser aceptables en relación al tipo de proceso a controlar y al valor del periodo de muestreo.

Un sistema empotrado tiene comunicación con una computadora vía puerto serial (USB o RS232C) para descargar programas y envío bidireccional de información como parámetros, datos y, en general para propósitos de diagnóstico e interpretación de resultados; pero no cuenta con accesorios del tipo multimedia, ni teclado, monitores o displays, discos duros, mouse, etcétera, debido a que está dedicado exclusivamente a ejecutar la aplicación del proceso físico en tiempo real. En algunos casos, sólo tiene simples exhibidores de cristal líquido (LCD) para desplegar comandos en texto.

En contraste, una computadora personal es multitareas, recibe e-mails, interacciona con internet, reproduce música y video, contiene numerosos paquetes de cómputo de texto, programas CAD (Computer Aided Design), diversos lenguajes de programación, cantidad suficiente de memoria; además, tiene un conjunto de accesorios multimedia, impresoras, teclado, monitor táctil e interactivo, etc. Sin embargo, el sistema empotrado no tiene todos esos accesorios, ni recursos, ya que de manera exclusiva atiende la tarea de llevar a cabo el procesamiento numérico para automatizar la planta. Por lo tanto, una computadora puede ser considerada como un conjunto de sistemas empotrados que atienden tareas específicas o bien definidas.

Sistema empotrado

Un sistema empotrado es definido como un sistema mínimo digital de propósito específico, cuya plataforma electrónica depende de un microcontrolador con el lenguaje de programación y periféricos adecuados para realizar una tarea exclusiva de automatización en tiempo real. Algunos ejemplos de sistemas empotrados se enlistan a continuación:



Reproductores de MP3.



Sistemas de navegación en aeronaves.



Control de tránsito en líneas ferroviarias.



Consolas de control de robots manipuladores industriales.

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

Fernando Reyes Cortés • Jaime Cid Monjaraz

6 Introducción



1.3 Sistema empotrado Arduino

E la sistema empotrado Arduino básico utiliza como unidad central de procesamiento (CPU) un microcontrolador de la familia ATMEL (ATmega 48PA/ 88PA/168PA/328PA), memoria flash y un conjunto de circuitos periféricos para adquirir datos, procesamiento y control de la información, así como comunicación serie con la computadora para enviar y recibir comandos. El diagrama a bloques del sistema electrónico básico de las tarjetas Arduino se muestra en la figura 1.1. Contiene memoria estática RAM (Random Access Memory) para almacenar variables y parámetros del programa (sketch), memoria tipo flash, que por sus características de alta velocidad, densidad y rápido acceso, se utiliza para almacenar código de máquina y la ejecución eficiente del sketch. Mientras que, Arduino Due utiliza el microcontrolador ARM CortexM3 con 32 bits y 84 MHz, y la tarjeta Galileo emplea el microcontrolador Intel Quark Soc X1000, 32 bits Pentium 400 MHz.

El sistema electrónico también incluye, memoria EEPROM (E²PROM Electrically Erasable Programmable Read-Only Memory) para almacenar un pequeño sistema operativo que se encarga de inicializar la tarjeta electrónica, configura puertos I/O en alta impedancia, convertidores analógico/digital en cero Volts, programa circuitos temporizadores o timers para generar la señal del tiempo real y pulsos de sincronía para el adecuado funcionamiento de los periféricos internos e interfaces electrónicas con el mundo exterior. Además, establece el protocolo de comunicación serie con la computadora para el intercambio de información. En esta memoria, también se encuentra el programa cargador que recibe el código de máquina que envía la computadora por medio del ambiente de programación Arduino.

El sistema electrónico Arduino interacciona con el mundo exterior a través de circuitos de interface como convertidores analógico/digital y digital/analógico de 10 y 12 bits, puertos de entrada/salida (I/O), señales de modulación de ancho de pulso (PWM), etc. La base de tiempo que permite obtener tiempo-real en control digital se realiza por medio de una señal periódica cuadrada programada en los timers. Tiempo real es una característica deseable en todo microcontrolador dedicado al control de sistemas físicos, cuyas aplicaciones pueden ser útiles en automatización de procesos térmicos de hospitales, hornos industriales, fabricación de dispositivos semiconductores, planeación de trayectorias de robots manipuladores, salas de cirugía, control de tráfico vehicular, etc.

Otra característica del sistema empotrado es el manejo de interrupciones para la adquisición de datos, procesamiento y control del proceso; los microcontroladores ATMEL contienen un dispositivo controlador de interrupciones que permite procesar solicitudes de interrupción interna o externa como pueden ser la lectura de algún sensor, convertidor analógico/digital, así como enviar y recibir información por comunicación serie en cualquier momento, con la finalidad de que el microcontrolador se dedique exclusivamente al control del proceso físico.

ALFAOMEGA ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA FERNANDO REYES CORTÉS • JAIME CID MONJARAZ

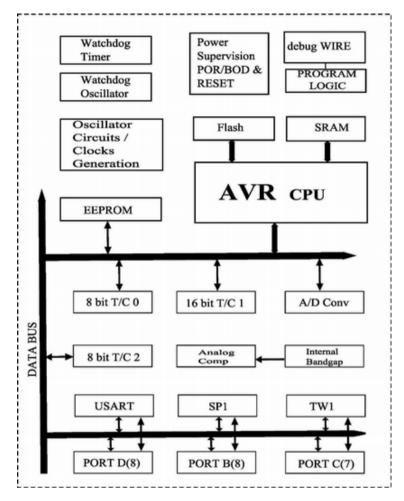


Figura 1.1 Diagrama a bloques del sistema Arduino básico.

Los componentes principales que integran al sistema Arduino son un ambiente de programación (IDE) orientado a la implementación práctica de esquemas de automatización para procesos físicos y la tarjeta electrónica. Las características principales de estos componentes se listan a continuación:



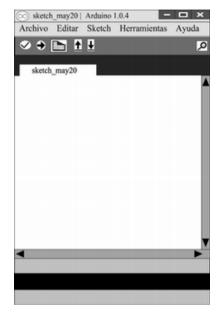
Un ambiente de programación (IDE) orientado al desarrollo de programas denominados sketchs para llevar a cabo aplicaciones de automatización en ciencias exactas e ingeniería. Este paquete se instala en la computadora personal, cuya ventana principal se muestra en la figura 1.2. Contiene un conjunto de comandos y funciones que facilitan el proceso de compilación, descarga de código y ejecución del sketch en la tarjeta Arduino.

El paquete de cómputo Arduino incluye drivers para su correcta operación en los sistemas operativos Windows, Linux y MaC OS, un editor de texto para edición de sketchs o programas

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

Fernando Reyes Cortés • Jaime Cid Monjaraz

8 Introducción



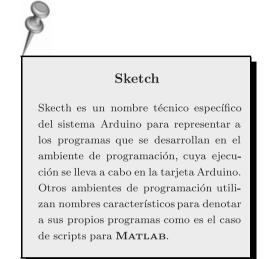


Figura 1.2 Ambiente de programación Arduino (IDE).

del usuario, compilador de los lenguajes C y C++ que detecta errores de sintaxis y la correspondiente fase de generación del código de máquina del programa sketch que se envía a la tarjeta electrónica por medio de un comando que emplea comunicación USB (descarga el código de máquina para su ejecución en la tarjeta electrónica). De esta forma, no requiere de un programador de código, como normalmente sucede con algunos microcontroladores y PIC's.

Una vez que se encuentra en ejecución el programa en la tarjeta Arduino, el ambiente de programación mantiene en todo momento la comunicación serie entre la computadora y la tarjeta electrónica para intercambio de información (recibir/transmitir datos).

El paquete de cómputo también dispone de una enorme cantidad de ejemplos didácticos, proporciona código fuente para ilustrar la forma de controlar servomotores, motores a pasos, comunicación serie, adquisición de datos de sensores, asistencia en línea de aspectos técnicos de programación o del hardware de la tarjeta electrónica (problemas frecuentes de implementación práctica) y actualización permanente del software.



Una ventaja importante del sistema Arduino es que una vez descargado el sketch a la tarjeta electrónica, éste queda en forma residente, ejecutándose sin importar que se desconecte la comunicación USB, apagar la computadora o la tarjeta. Al conectar nuevamente los sistemas, la comunicación se restablece automáticamente, con la ejecución del último sketch cargado.

Alfaomega

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

FERNANDO REYES CORTÉS • JAIME CID MONJARAZ

Una característica importante del sistema Arduino es que el programa sketch quedará residente en la memoria flash no volátil, tiene la ventaja que cuando se desconecta la alimentación de la tarjeta y al momento de conectar nuevamente la alimentación, el programa se ejecutará.



El paquete completo de cómputo Arduino se encuentra disponible en forma gratuita en el siguiente sitio Web:

www.arduino.cc

Las tarjetas electrónicas Arduino pueden ser de diversos modelos, dependiendo del tipo de microcontrolador ATMEL que utilicen y de las características de los circuitos periféricos de soporte. Los principales modelos son: UNO, Leonardo, Duemilanove, Mega, Mini, Diecimila, Due, Yune, Wi-Fi, Ethernet y Galileo.

El ambiente de programación contiene los comandos necesarios para editar, corregir errores de sintaxis del lenguaje C, además de compilar y generar el código de máquina del tipo de microcontrolador utilizado de la familia ATMEL, así como la transferencia de ese código hacia la tarjeta para su ejecución (loader). La figura 1.3 muestra el diagrama esquemático de interacción del ambiente de programación con la tarjeta electrónica.



1.3.1 Arquitectura abierta del sistema Arduino

La filosofía de trabajo del sistema Arduino es **arquitectura abierta**, consiste en adaptar a la medida la programación e implementación de una aplicación en función de las características específicas del proceso físico, es decir, es la forma de programar e implementar la automatización requerida de un proceso particular. La parte clave de la arquitectura abierta es el ambiente de programación con el conjunto de comandos y funciones que proporcionan al usuario todas las herramientas necesarias para diseñar y desarrollar la aplicación sin pérdida de tiempo. El sistema electrónico contiene los componentes y circuitos necesarios para acoplar una enorme variedad de aplicaciones de ciencias exactas y del área de ingeniería. Cuando el proceso demanda una aplicación especializada, la interface electrónica adicional puede ser fácilmente acoplada a la tarjeta.

En el ambiente de programación se realiza la edición del sketch, compilación y depuración de errores de sintaxis de acuerdo con las reglas gramaticales de los lenguajes C y C++; así como la generación automática del código de máquina del correspondiente tipo de microcontrolador de la familia ATMEL y la ejecución del sketch es realizada por el microcontrolador. Información de interés al usuario como variables de estado y errores de posición son transmitidas a la computadora para presentarla en formato numérico o gráfico, esto último en MATLAB o LabVIEW; admite modificación en línea de parámetros, sin suspender la ejecución del sketch.

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

Fernando Reyes Cortés • Jaime Cid Monjaraz

10 Introducción

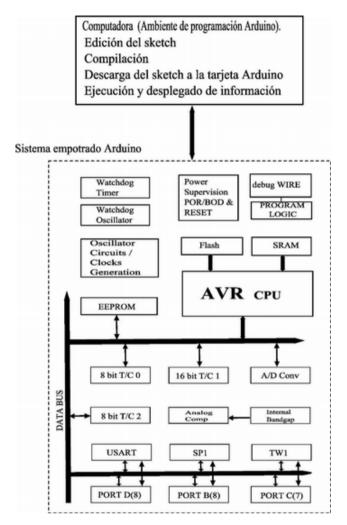


Figura 1.3 Interconexión de trabajo del ambiente de programación con la tarjeta electrónica.

En la figura 1.4 se ilustran los pasos secuenciales requeridos para llevar a cabo la ejecución de un sketch en la tarjeta Arduino y desplegar información requerida por el usuario en la computadora. Este proceso inicia con la edición del sketch en el ambiente de programación, es decir, se implementa en lenguaje C la aplicación específica para automatizar la planta. En este punto es importante considerar que en el momento de compilar es posible que se presenten diversos errores de sintaxis, en este caso, el ambiente de programación indica la fuente del error, para que a través del editor de texto se corrigen, compilando una vez más hasta eliminar todos los errores gramaticales, entonces la generación del código de máquina para el microcontrolador ATMEL se realiza de manera

Alfaomega

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

automática. A partir de este momento se pude descargar el código de máquina del sketch en la tarjeta electrónica para su ejecución usando el comando cargar código.

Cuando el sketch se encuentra en ejecución en la tarjeta Arduino, es posible enviar/recibir información. Por ejemplo, la tarjeta Arduino contiene un pequeño programa operativo almacenado en la memoria EEPROM con subrutinas especiales para manejar solicitudes de comunicación (monitor serial), es decir, establece el protocolo para enviar información de interés del usuario a la computadora como variables, banderas de control y valores de parámetros; también acepta información para modificar el valor numérico de algún parámetro del sketch.

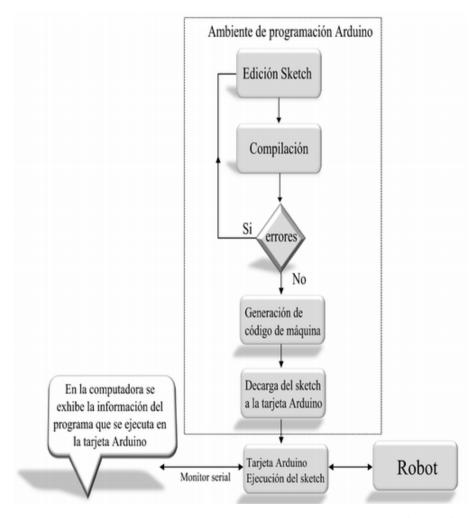


Figura 1.4 Pasos secuenciales para ejecutar un sketch en la plataforma Arduino.

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

FERNANDO REYES CORTÉS • JAIME CID MONJARAZ

12 Introducción

Aspectos históricos del sistema Arduino

La figura 1.5 muestra al equipo fundador del sistema Arduino; de izquierda a derecha están: David Cuartielles, Gianluca Martino, Tom Igoe, David Mellis, y Massimo Banzi.

El nombre Arduino corresponde al lugar que frecuentaban los integrantes del equipo fundador en el pueblo de Ivream ubicado al norte de Italia (Bar di Re Arduino, llamado así en honor al Rey Arduino, quien vivió en el año 1002). El sistema Arduino nació en 2005 como una necesidad de subsistir por el cierre del Instituto de Diseño Interactivo IVREA.

La idea original fue concebido por los pioneros, el italiano Massimo Banzi y el español David Cuartielles como un sistema académico en arquitectura abierta (open hardware), para no tener que enfrentar problemas legales de uso, a bajo precio y del tipo plug and play.

En el proyecto participaron varios tesistas de ingeniería, realizando la programación de diversos bloques, comandos y funciones del sistema, entre los que sobresale Hernando Barragán (colombiano), quien desarrollo la plataforma *Wiring* para programar el microcontrolador.

La primera producción de tarjetas Arduino fue de alrededor de 300 unidades y se las proporcionaron a los alumnos del Instituto IVRAE para realizar proyectos académicos; en aquel entonces, las ganancias fueron de un euro por tarjeta electrónica.

Dentro de las primeras aplicaciones Arduino se encuentran la del reloj despertador/alarma. Posteriormente, el profesor Tom Igoe (considerado el padre de la computación física) se unió al proyecto para desarrollarlo a grandes escalas.

Hoy en día, el sistema Arduino se utiliza en proyectos de investigación, desarrollo tecnológico y docencia de las principales universidades de todo el mundo.



Figura 1.5 Equipo fundador Arduino

Alfaomega

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA



Massimo Banzi Ingeniero italiano

Massimo Banzi, nació en Italia, recibió el grado de ingeniería en ITIS Enrico Fermi Desio y fue cofundador del sistema Arduino. Ingeniero y diseñador de sistemas electrónicos en arquitectura abierta, trabajó como consultor para clientes de las compañías: Prada, Artemide, Persol, Whirlpool, V&A Museum y Adidas. Estuvo trabajando 4 años en el Instituto de Diseño Interactivo IVREA como profesor asociado. Massimo es fundador de FabLab en Torino y autor del libro "Getting Started with Arduino" editado por O'Reilly, 2009.

Actualmente es profesor de la Academia de SUPSI y Domus.



Ejemplos prácticos con Arduino

En el sitio Web del libro, el lector puede consultar diversas aplicaciones del sistema Arduino para las área de robótica e ingeniería mecatrónica. Están disponibles manuales, documentación técnica de programación e interfaces electrónicas sobre aplicaciones de control de servomotores, procesos térmicos, motores a pasos, acondicionamiento de señales y adquisición de datos de sensores de fuerza, potenciómetros, interruptores mecánicos, direcciones electrónicas de interés, etc.

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

Fernando Reyes Cortés • Jaime Cid Monjaraz

14 Introducción



1.4 Resumen

E la sistema Arduino se ha convertido en una plataforma de programación y automatización con diversas aplicaciones en ingenierías y ciencias exactas, paulatinamente ha incursionado en universidades, institutos de investigación e industria, particularmente en docencia, hoy en día, representa una sofisticada herramienta que facilita el proceso de enseñanza-aprendizaje de cursos de matemáticas, física, robótica, mecatrónica, instrumentación electrónica, automatización, control digital, entre otros cursos más.

Arduino es un sistema empotrado dedicado a realizar aplicaciones de automatización de procesos físicos, por medio de una plataforma electrónica que emplea un microcontrolador de la familia ATMEL y a través de un ambiente de programación orientado a la implementación de aplicaciones de ingeniería (robótica, mecatrónica, informática, industrial), física y matemáticas; este paquete de cómputo proporciona al usuario de todas las herramientas necesarias para realizar con facilidad la automatización de procesos físicos utilizando lenguaje C, es decir, simplifica notablemente los aspectos prácticos de implementación, modelado, caracterización y calibración de sensores, procesamiento y control de variables de estado, envío de señales y comandos, descarga de programas e intercambio de información usando comunicación USB con la computadora y también enlace con otros paquetes de cómputo tales como MATLAB, LabVIEW, etc.

Hay varios modelos de tarjetas Arduino que tienen aplicaciones desde adquisición y lectura de datos, acoplamiento de sensores de temperatura, presión, fuerza, calibración de señales no lineales, algoritmos de procesamiento digital de señales, métodos numéricos, control de servomotores, robots móviles, brazos robots industriales, comunicación Bluetooth, Wi-Fi, Ethernet, etc.

Documentación técnica de aplicaciones, foros y actividades académicas, manuales de operación, programación, clases de tarjetas electrónicas y sociedades Arduino. Actualización permanente del ambiente de programación se encuentran en:

www.arduino.cc

Alfaomega

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

FERNANDO REYES CORTÉS • JAIME CID MONJARAZ

1.5 Referencias selectas



E N la actualidad hay una cantidad enorme de literatura que describe al sistema Arduino, este se debe a la enorme popularidad de esta plataforma electrónica; a continuación se recomienda al lector un conjunto de referencias bibliográficas claves para ampliar su óptica sobre el proceso de automatización de procesos físicos usando el sistema Arduino.

- Massimo Banzi "Getting started with Arduino". First edition, 2009. Published by O'Reilly Media, Inc.
- Jonathan Oxer and Hugh Blemings "Practical Arduino: Cool projects for open source hardware". Appres, 2009.
- Casey Reas and Ben Fry "Getting started with Processing". First edition, 2010. Published by O'Reilly Media, Inc.
- Michael McRoberts "Beginning Arduino". Apress, 2010.
- Alasdair Allan "iOS Sensor apps with Arduino". First edition, 2011. Published by O'Reilly Media, Inc.
- John David Warren, Josh Adams and Harald Molle "Arduino Robotics". Apress, 2011.
- Mike Riley "Programming your home: Automate with Arduino, Android and your computer". The Pragmatic Bookshelf, 2012.
- Tom Igoe "Getting started with RFID". First edition, 2012. Published by O'Reilly Media, Inc.
- "Embedded Systems: Building and programming embedded devices". Disponible en forma gratuita en Wikibooks: http://en.wikibooks.org, este libro fue creado por autores voluntarios.

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

Fernando Reyes Cortés • Jaime Cid Monjaraz

16 Introducción

En el sitio Web del sistema Arduino el lector puede descargar en forma gratuita el paquete de cómputo completo para realizar aplicaciones en ingeniería y ciencias exactas:



www.arduino.cc

Los detalles y aspectos históricos del sistema Arduino y sus pioneros fueron consultados en la revista spectrum de IEEE http://spectrum.ieee.org/ (ver la siguiente dirección):



http://spectrum.ieee.org/geek-life/hands-on/the-making-of-arduino



1.6 Problemas propuestos

E STA sección contiene una serie de problemas conceptuales con la finalidad de hacer reflexionar al lector sobre las características técnicas y los aspectos tecnológicos del sistema Arduino.

- 1.6.1 ¿Qué es un sistema empotrado?
- 1.6.2 Describa las características principales de un sistema empotrado.
- 1.6.3 ¿Qué es un sistema en tiempo real?
- 1.6.4 ¿Cuáles son las características fundamentales de la arquitectura abierta del sistema Arduino?
- 1.6.5 ¿Por qué se utiliza el lenguaje C en el sistema Arduino?
- 1.6.6 ¿Por qué se ha posicionado rápidamente Arduino en el mercado mundial?
- 1.6.7 Mencione al menos cinco paquetes de cómputo que se pueden enlazar por medio del puerto USB con sistema Arduino.
- 1.6.8 ¿Qué componentes forman la plataforma electrónica del sistema Arduino?
- 1.6.9 Describa cinco aplicaciones de automatización del sistema Arduino.

Alfaomega

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA



Instalación y puesta a punto del sistema Arduino



- 2.1 Introducción
- 2.2 Instalación
- 2.3 Ambiente de programación Arduino
- 2.4 Puesta a punto
- 2.5 Resumen
- 2.6 Referencias selectas
- 2.7 Problemas propuestos

Competencias

Presentar el procedimiento de instalación y puesta a punto del sistema Arduino. Asimismo, en este capítulo se incluye un manual de usuario sobre el ambiente de programación con todas las funciones y herramientas para desarrollar aplicaciones de ingeniería robótica y mecatrónica.

Desarrollar habilidades en:

- Descripción del sistema Arduino (software libre y hardware en arquitectura abierta).
- Funcionalidad y características del ambiente de programación.
- Comprobación de la puesta a punto del ambiente de programación Arduino.
- Desarrollo de programación a través de ejemplos didácticos.

2.1 Introducción 19

2.1 Introducción



La instalación y puesta a punto de la plataforma Arduino son procesos claves para la programación de aplicaciones en ingeniería y ciencias exactas. Representan el punto de arranque y al mismo tiempo significan la confianza que el usuario pueda tener en este sistema de programación para desarrollar aplicaciones.

La ingeniería robótica y mecatrónica se han convertido en protagonistas de las transformaciones realizadas en las últimas décadas, además de ser áreas estratégicas y prioritarias en el progreso de la sociedad, por lo cual, son elementos fundamentales para el desarrollo del país, ya que resultan factores imprescindibles del avance nacional, elevando los niveles de competitividad en función de los indicadores requeridos por organizaciones internacionales de un ámbito globalizado. Las sociedades que poseen la capacidad necesaria para ser vanguardistas, diseñar, elaborar y producir a gran nivel tecnología de punta llegan a mejorar incluso la calidad de vida de sus habitantes.

Para que un país alcance un rápido desarrollo, la investigación científica y el desarrollo de tecnologías deben ser actividades nacionales preponderantes, a tal grado que el país ubique en el centro de su quehacer la formación de recursos humanos con un perfil científico-tecnológico, mediante el desarrollo de prototipos mecatrónicos, así como la generación y aplicación de conocimientos. En este sentido, en el mundo hay una tendencia hacia la modernización de procesos mediante la síntesis y la sinergia de productos con componentes que tradicionalmente habían sido tratados de manera independiente por su diversa naturaleza, aunque hoy, esos conceptos han sido integrados por la mecatrónica y robótica.

Si se considera el factor tecnológico dentro del contexto clave y estratégico que ocupa la ingeniería a nivel mundial, el desarrollo de aplicaciones de automatización puede darse de manera inmediata usando sistemas empotrados (embedded systems); particularmente, la plataforma Arduino fue pensada para desarrollar aplicaciones de ingeniería en forma simple y rápida, debido a que proporciona al usuario todas las herramientas para diseñar aplicaciones sin requerir componentes adicionales como son los programadores de código de algunos microcontroladores y PICs (Peripheral Interface Controllers). Arduino ofrece ventajas importantes en el campo de la docencia, pero también facilita el proceso de aprendizaje.

El sistema Arduino contiene un ambiente de programación (IDE) orientado al desarrollo de aplicaciones de automatización de procesos físicos, con comandos y funciones integrados en el ambiente, lo que facilita los pasos secuenciales para escribir aplicaciones en los lenguajes C/C++, así como la ejecución del sketch sobre una tarjeta electrónica.

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

FERNANDO REYES CORTÉS • JAIME CID MONJARAZ

En este capítulo se describen los pasos necesarios para instalar el paquete de cómputo Arduino en un computadora personal, así como para habilitar los controladores o drivers para que funcionen correctamente los diversos modelos de tarjetas electrónicas usando el puerto USB del sistema operativo Windows. La puesta en operación del sistema significa que éste se encuentra listo para programar aplicaciones de ingeniería mecatrónica y robótica, con la ejecución correcta del sketch en la plataforma electrónica, así como para el intercambio de información con la computadora. Esta etapa se ilustra por medio de ejemplos didácticos que describen los pasos previos para la compilación del sketch y la descarga del código de máquina hacia la tarjeta electrónica.

También en este capítulo, se destina una sección específica para describir los comandos y funciones del ambiente de programación con la finalidad de proporcionar al lector un manual con la información relevante para desarrollar aplicaciones de automatización de procesos físicos.

El sistema Arduino está compuesto por la plataforma de programación (paquete de cómputo) y una tarjeta electrónica con las siguientes características:



El paquete de cómputo Arduino es un ambiente integrado de programación multiplataforma (Linux, MacOS X y Windows) conocido como **IDE** (*Integrated Development Environment*), el cual es gratuito y libre. Este paquete de cómputo fue escrito en Java y Processing, está listo para ser descargado del siguiente sitio Web:

www.arduino.cc



La tarjeta electrónica incorpora un microcontrolador programable con circuitos periféricos para acoplar y acondicionar electrónicamente señales de sensores (analógicos y digitales), así como el envío de comandos de control. Hay varios modelos de tarjetas electrónicas, lo cual depende del tipo de microcontrolador ATMEL utilizado y de la clase de características técnicas de sus componentes electrónicas (véase capítulo 3 "Plataforma electrónica").



2.2 Instalación

E l proceso de instalación es un proceso metodológico que consiste en llevar a cabo un conjunto de pasos secuenciales para obtener un funcionamiento correcto del ambiente de programación con sus herramientas, funciones y comandos (edición, compilación, monitoreo serial, librerías, descarga de código de máquina, etc.), así como para permitir que el canal de comunicación USB opere adecuadamente con los diferentes modelos de tarjetas electrónicas.

A continuación se describen los pasos necesarios para instalar el sistema Arduino:

Alfaomega Arduino. Aplicaciones en Robótica y Mecatrónica

2.2 Instalación 21



Descargar la última versión del paquete de cómputo Arduino del sitio Web:

www.arduino.cc



La dirección específica de descarga se encuentra en la siguiente liga:

http://arduino.cc/en/Main/Software

Dentro del proceso de instalación, automáticamente el paquete de cómputo detecta el tipo de idioma que tiene la computadora. También tiene la opción de configuración **Preferencias** para el idioma castellano (véase página 28).



Seleccionar el sistema operativo (Linux, MacOS X o Windows). Por ejemplo, para el sistema operativo Windows, la descarga inicia con un archivo **arduino-00XX-windows.zip**, donde **XX** representa la última versión del paquete de cómputo y también anexa la clase de sistema operativo. Este archivo contiene la siguiente información:



Un conjunto de carpetas denominadas: tools, reference, lib, java, hardware, examples, drivers, librerías (rxtxSerail.dll, revisions.txt, libusb0.dll, cygwin1.dll, cygiconv-2.dll) y la aplicación **arduino.exe**.



Cuando finalice la descarga del sitio Web, habrá que descomprimir el archivo **arduino-00 XX-windows.zip**; durante la descompresión puede indicar la trayectoria o el nombre de la carpeta, por ejemplo, **c:\Arduino**\ aquí será alojado el sistema Arduino.



Una vez finalizada la descompresión del archivo descargado, habrá que abrirse la carpeta donde se alojó toda la información del paquete de cómputo, así como ubicar y ejecutar la aplicación **arduino.exe**, el cual se distingue por el icono ••• Conserve la estructura original de carpetas y subdirectorios instalados.



Cuando estén efectuadas las fases anteriores, entonces se requiere contar con un modelo de tarjeta electrónica Arduino, por ejemplo el modelo UNO y con el cable específico para comunicación USB, como se indica en la figura 2.1.



2.2.1 Instalación de drivers de las tarjetas Arduino

La primera vez que conecte un modelo de tarjeta Arduino al puerto USB, los *drivers* descargados deberán instalarse automáticamente para los sistemas Windows Vista y Windows 7, aparecerá una ventana similar a la indicada en la figura 2.2. Ahora, ya está listo para programar.

Página de consulta sobre el proceso de instalación Arduino

En la siguiente página el usuario puede consultar el proceso de instalación del ambiente de programación de la plataforma Arduino para el sistema operativo Windows:

http://arduino.cc/es/Guide/Windows

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

Fernando Reyes Cortés • Jaime Cid Monjaraz



Figura 2.1 Tarjeta Arduino UNO y cable estándar para puerto USB.

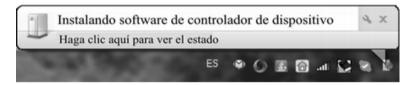
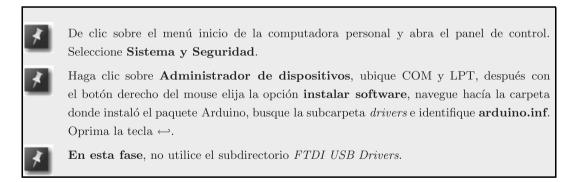


Figura 2.2 Instalación automática de drivers de Arduino.

No obstante, en algunas ocasiones la instalación de los *drivers* no es inmediata, además de que es posible se presenten algunos problemas técnicos. En tal caso, se requiere llevar a cabo el siguiente procedimiento:



Drivers con firma no reconocida

Otro problema frecuente que suele presentarse al momento de instalar los *drivers*, es que el sistema operativo Windows no admite la instalación de drivers con firma no reconocida. En este caso se recomienda utilizar el siguiente procedimiento:

Alfaomega

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

2.2 Instalación 23

Deshabilitar el uso obligatorio de controladores no firmados

Para Windows 8 y Windows 8.1, lleve el cursor del mouse hasta la esquina superior derecha, haga clic en el icono de **configuración** (el icono con forma de engranes del sistema operativo).

Haga clic en Cambiar configuración de PC, seleccione Uso general y ahora en la opción de Inicio avanzado, de clic en Reiniciar ahora.

Después que se reinicializa la computadora, aparecerá un menú titulado Elegir una opción, de clic en Opciones avanzadas y luego en Configuración de inicio, de clic nuevamente en reinicio, elegir opción 7 "Deshabilitar el uso obligatorio de controladores no firmados".

Conecte la tarjeta Arduino y proceda tal y como se indica al inicio de la subsección 2.2.1 "Instalación de drivers de las tarjetas Arduino" (véase página 21).

Para los sistema operativos Windows 7 y Vista, reinicie compu-F8 tadora oprima varias veces la tecla (función: F_n 8), "Deshabilitar el uso obligatorio de controladores no firmados", entrar el sistema operativo, conecte la tarjeta Arduino y proceda como en el punto inmediato anterior.

De manera similar, el proceso de instalación del paquete Arduino en el sistema operativo MacOS X consiste en los siguientes pasos secuenciales:

- 1) Descargar la última versión del software Arduino para el sistema operativo MacOS X directamente de la página: http://arduino.cc/en/Main/Software.
- 2) Cuando finalice la descarga, descomprima el archivo zip que generalmente contiene un nombre asociado al número de versión de la plataforma Arduino y la clase de sistema operativo a emplear. Por ejemplo, arduino-num_versión-macosx.zip. Durante la descompresión puede indicar la trayectoria o el nombre de la carpeta donde sería alojado el sistema Arduino.
- 3) Una vez terminada la descompresión del archivo descargado, de manera automática se genera el icono Arduino para trabajar inmediatamente.

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

Fernando Reyes Cortés • Jaime Cid Monjaraz



2.3 Ambiente de programación Arduino

E la ambiente de programación Arduino permite edición de programas, desarrollo, compilación, depuración de errores de programación, así como la descarga del código de máquina con los modelos de tarjetas electrónicas, cuyas plataformas giran alrededor de un tipo de microcontrolador ATMEL. También permite intercambiar información de datos o comandos por medio del puerto USB. El ambiente de programación representa la interface de desarrollo entre el usuario y las tarjetas Arduino para realizar de manera amigable aplicaciones específicas en ingeniería robótica, control, automatización, mecatrónica, sistemas e instrumentación electrónica, física y matemáticas.

Una vez instalado el software de descarga Arduino, la primera vez que se utilice el ambiente de programación es necesario ir a la carpeta de instalación que definió el usuario, por ejemplo c:\Arduino\, y ejecutar el programa arduino.exe, el cual se identifica por el icono . En esta fase inicial, y por comodidad es recomendable que el usuario coloque en el escritorio de su computadora una trayectoria directa de ejecución del ambiente de programación. La manera de realizarlo es oprimiendo el botón derecho del mouse y dar clic sobre el programa arduino.exe, seleccionando la opción Crear acceso directo.

La figura 2.3 es la reproducción de la ventana principal del ambiente de programación Arduino, donde en la parte superior izquierda se muestra el nombre inicial del archivo que genera y la versión del ambiente. Posteriormente se ilustra el menú principal con las opciones de que dispone la interface de programación tales como Archivo, Editar, Sketch, Herramientas y Ayuda. Los nombres de los programas Arduino se denominan sketch y normalmente se crean de acuerdo con la fecha en que son editados, por ejemplo sketch_may20 corresponde al 20 de mayo del año actual (la extensión que usa el ambiente de programación en los archivos sketch es .ino). El usuario puede renombrar el archivo sketch con cualquier otro nombre con símbolos alfanuméricos; sin insertar alguna extensión, en forma automática se anexa al nombre del archivo .ino, rasgo distintivo del sistema Arduino.

El ambiente de programación Arduino trae una barra de herramientas con botones que activan comandos de fácil acceso para la compilación y descarga de programas a las tarjetas Arduino, para así mantener la comunicación serial entre la tarjeta y la computadora. La forma de activar los comandos es por medio del botón derecho del mouse. A continuación se listan los principales comandos para el desarrollo de sketchs.



Para verificar y compilar la sintaxis del programa del usuario de acuerdo a la gramática del lenguaje C/C++ se utiliza el icono \bigcirc .



La descarga del código de máquina generado del proceso de compilación del sketch hacia la tarjeta Arduino se lleva a cabo con el puerto USB por medio del icono .

Alfaomega

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA



Figura 2.3 Ventana principal del ambiente de programación Arduino.

- El icono genera un nuevo archivo sketch, cuyo nombre depende de la fecha actual, y para distinguirlo de otros programas previos con la misma fecha, se le añadirá al final del nombre una letra del alfabeto castellano; por ejemplo: sketch_may20a se distingue del programa anterior sketch_may20 por la letra agregada (en este caso la letra a), aunque ambos programas corresponden al 20 de mayo del año en curso.
- Para abrir un programa sketch previamente existente o generado se utiliza el comando 🟦
- Para guardar o salvar programas sketch se emplea el comando 🛨.
- El monitor serial permite exhibir en una ventana información de variables o comandos de la ejecución del programa del usuario en la tarjeta Arduino; por lado tenemos que por medio del icono se activa la visualización de esa información. Esta misma opción viene en el menú Herramientas o se obtiene de manera compacta oprimiendo simultáneamente las teclas Ctrl+Mayúsculas+M (véase la sección 2.4.2).

El ambiente de programación Arduino inicia directamente en modo de edición, lo que permite al usuario escribir instrucciones de programación sobre el área blanca que muestra la figura 2.3 (consola de texto). El editor de texto contiene barras navegadoras para desplazar el archivo en formas horizontal y vertical. Por debajo de la barra de deslizamiento horizontal se encuentran dos franjas destinadas a mostrar los mensajes del proceso de compilación y descarga de código a la tarjeta Arduino. Observe que en la parte inferior izquierda de la ventana principal del ambiente se exhiben el número de columna del editor de texto y el tipo de conexión para la comunicación USB.

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

Fernando Reyes Cortés • Jaime Cid Monjaraz



2.3.1 Menú Archivo

EL menú **Archivo** contiene un conjunto de comandos para crear nuevos programas (**Ctrl+N**), abrir (**Ctrl+O**), cerrar (**Ctrl+W**), guardar o salvar (**Ctrl+S**), guardar como (**Ctrl+Mayúsculas+S**) e imprimir archivos sketch (**Ctrl+P**). Un aspecto importante desde el punto de vista didáctico es que el software de instalación ya viene con un número importante de ejemplos que describen a detalle la forma de usar puertos I/O, convertidores analógico/digital y digital/analógico, comunicación serial con la computadora, memoria, etc. Desde el menú **Archivo** se puede acceder a todos los ejemplos ilustrativos por medio de la opción **Ejemplos**.

El menú **Archivo**, también permite descargar código de máquina del programa de usuario hacia la tarjeta Arduino (comando **Cargar** que se activa con la combinación de teclas **Ctrl+U**), previo proceso de compilación y configuración de las preferencias de funcionamiento del ambiente Arduino (**Ctrl+**,). Para finalizar la sesión de trabajo, se utiliza la opción **Salir**, con la que se suspende la ejecución del ambiente de programación (oprimir simultáneamente las teclas **Ctrl+Q**); en este punto, es necesario resaltar que el sketch que se encuentra ejecutando en la tarjeta electrónica no se suspende, continuará en ejecución autónoma (mientras tenga alimentación eléctrica), aun cuando el ambiente de programación ya esté cerrado. La figura 2.4 muestra todas las opciones con las que cuenta el menú **Archivo**.

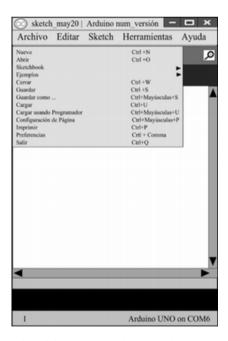


Figura 2.4 Menú Archivo del ambiente de programación Arduino.

Alfaomega

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

FERNANDO REYES CORTÉS • JAIME CID MONJARAZ

Opción Sketchbook

El ambiente de programación utiliza **sketchbook** para indicar el lugar específico donde radican o residen programas o sketchs. Se habilita desde el menú **Archivo**, opción **Sketchbook**. Cuando se ejecuta por primera vez el ambiente de programación Arduino, automáticamente se creará un directorio para sketchbook, en consecuencia, la trayectoria o localidad de ese directorio podrán ser visualizada con la opción **Preferencias** ubicada en el menú **Archivo** o simplemente oprima simultáneamente las teclas **Ctrl** más **coma** (**Ctrl+**,).

Opción Ejemplos

Como parte del menú **Archivo** se encuentra la opción **Ejemplos**, que contiene una colección interesante de ejemplos didácticos donde se muestra la forma de programación para diferentes aplicaciones.

La figura 2.5 presenta la clasificación de ejemplos disponibles que van desde muy básicos hasta especializados, con la finalidad de que usuario conozca el amplio universo de aplicaciones, y al mismo tiempo se familiarice con las características específicas de las tarjetas Arduino.

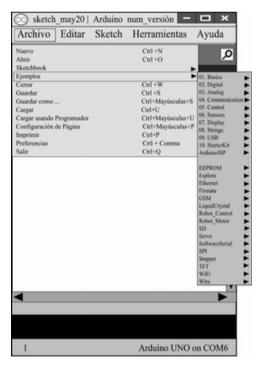


Figura 2.5 Ejemplos Arduino.

Los ejemplos detallan manejo de puertos digitales, entradas analógicas, acoplamiento y procesamiento de sensores, exhibidores (displays) con diodos LED o de cristal líquido (LCD), memoria EEPROM, control de servomotores de corriente directa, motores a pasos, comunicación vía USB, Bluetooth, Wi-Fi, Ethernet, etc.

El listado de ejemplos del ambiente de programación Arduino representa una poderosa herramienta documentada que proporciona una perspectiva amplia al usuario para llevar a cabo la implementación práctica de acondicionamiento, procesamiento y calibración de sensores analógicos y digitales, así como control de servomotores para robots móviles e industriales. El código fuente mostrado en cada uno de estos ejemplos sirve como base para extenderlo y adaptarlo a numerosas aplicaciones de automatización y de ingeniería robótica y mecatrónica.

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

Fernando Reyes Cortés • Jaime Cid Monjaraz

Opción Preferencias

El ambiente de programación Arduino está disponible en más de 30 idiomas diferentes de configuración; en primera instancia, el ambiente presenta los letreros de los menús de acuerdo con el idioma seleccionado por el sistema operativo de su computadora. Sin embargo, el idioma puede ser configurado por medio de la opción **Preferencias** del menú **Archivo** u oprimiendo en forma simultanea la combinación de teclas (**Ctrl+**,), con lo que se activa la ventana que se muestra en la figura 2.6.

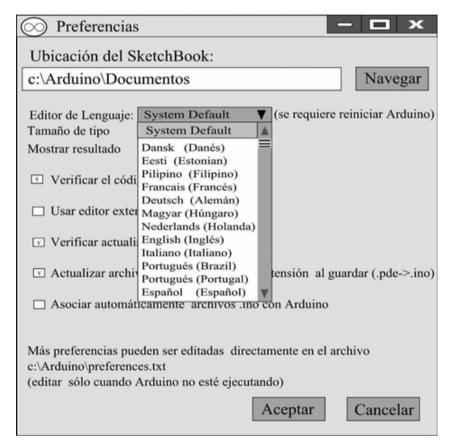
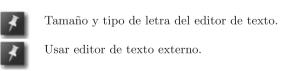


Figura 2.6 Ventana de trabajo de la opción Preferencias.

La ventana de trabajo de la opción **Preferencias** permite definir las siguientes características de funcionalidad del ambiente de programación:



Alfaomega Arduino. Aplicaciones en Robótica y Mecatrónica

- Al momento de iniciar el ambiente de programación Arduino, permite detectar la última actualización disponible en el sitio Web: https://www.arduino.cc bajar e instalar.
- Sirve para mostrar mensajes detallados (errores de sintaxis y advertencias o warnings) durante la fase de compilación y descarga de código a la tarjeta Arduino.
- Permite verificar el código después de la descarga a la tarjeta Arduino.
- Asocia automáticamente todos los archivos con extensión .ino con el ambiente de programación Arduino.
- A partir de la versión 1.0 del ambiente de programación Arduino, los programas se salvan como archivos con extensión .ino. Versiones previas a la 1.0 usan la extensión .pde. Para habilitar la conversión de archivos.pde a archivos.ino hay que hacer la indicación en el recuadro "Actualizar archivos de sketch a una nueva extensión al guardar (.pde->.ino)", forma en la cual pueden ser compilados y cargados a la versión 1.0, así como versiones posteriores.
- La dirección o trayectoria donde se ubica la carpeta Sketchbook se observan en la parte superior de la ventana. Cuando el usuario cambia la trayectoria de esta carpeta, para que tenga efecto esta modificación, hay que salirse del programa Arduino y ejecutarlo nuevamente.

Otras características de configuración del ambiente de programación Arduino, tales como opciones de compilación y descarga, tamaño de la sangría (tab), número de columnas y renglones del editor de texto, guardar o salvar archivos en forma automática, son factibles de ser modificados en el archivo preferences.txt.

El archivo **preferences.txt** se encuentra en la carpeta de instalación del sistema Arduino que el usuario previamente definió y que se despliega en la parte inferior de la ventana de **Preferencias**; por ejemplo **c:\Arduino\preferences.txt**, el cual puede ser abierto y modificado con cualquier editor de textos. La edición de las preferencias u opciones de configuración deberá realizarse cuando el ambiente de programación Arduino no esté en ejecución.

La tabla 2.1 muestra el resumen de comandos del menú **Archivo** que se pueden activar con el teclado, es decir, por medio de combinar determinadas teclas los comandos de este menú llegan a ejecutarse de manera instantánea.

Hay que tomar en cuenta, que la combinación de teclas para activar un comando es irrelevante que se encuentre la letra correspondiente en modo mayúscula o minúscula; por ejemplo, para generar un nuevo archivo por medio del comando **Sketch nuevo** (este comando se ubica dentro del menú de **Archivo**), el camino más corto, consiste en oprimir en forma simultanea las teclas **Ctrl N**, pero también puede oprimirse las teclas **Ctrl n**.

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

Fernando Reyes Cortés • Jaime Cid Monjaraz

Tabla 2.1 Comandos del menú Archivo.

| Descripción del comando | Combinación de teclas |
|---------------------------|-----------------------|
| Sketch nuevo | Ctrl+N |
| Abrir sketch | Ctrl+O |
| Cerrar sketch | Ctrl+W |
| Guardar sketch | Ctrl+S |
| Guardar como | Ctrl+Mayúsculas+S |
| Cargar programa | Ctrl+U |
| Cargar usando programador | Ctrl+Mayúsculas+U |
| Configuración de página | Ctrl+Mayúsculas+P |
| Imprimir | Ctrl+P |
| Preferencias | Ctrl+, |
| Salir | Ctrl+Q |

Nota importante

La combinación de teclas $\mathbf{Ctrl} + \mathbf{N}$ es una notación que se logra al oprimir primero la tecla de control \mathbf{Ctrl} y luego, sin liberar esa tecla, oprimir la tecla \mathbf{N} ; después de esto, hay que sostener ambas teclas hasta que se ejecute el comando. Lo anterior no significa oprimir en ningún momento el símbolo +. La tecla de la letra puede estar en modo minúscula o en modo mayúsculas, ya que para la ejecución del comando resulta irrelevante.

La notación Ctrl+Mayúsculas+S corresponde al comando Guardar como... La notación debe interpretarse como oprimir en primera instancia la tecla control Ctrl; luego, sin liberar esa tecla, oprimir la tecla de Bloq Mayúsculas y no soltar ninguna de las dos; por último, oprimir la tecla S. En ese momento, estarán oprimidas en forma simultánea las tres teclas y esperar a que se active el comando; resulta irrelevante si está la tecla en minúsculas o en mayúsculas.

Hay que mantener en mente, que esta regla se aplica para cualquier comando con una combinación similar de teclas.

Alfaomega

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

FERNANDO REYES CORTÉS • JAIME CID MONJARAZ

Menú Editar

El editor de textos integrado al ambiente de programación permite escribir sketchs en la consola de texto. Contiene, asimismo, funciones comunes de edición de programas tales como: copiar/pegar, buscar/remplazar y cortar texto (letras, palabras, párrafos o páginas enteras previa selección).

En modo **Edición** se pueden insertar y eliminar comentarios del lenguaje C/C++; los comentarios en un programa representan notas claves que facilitan la documentación técnica del código, y no aumentan la cantidad de bytes al momento de ejecutar al sketch en la tarjeta Arduino. Los comentarios pueden ser de una línea y generalmente se encuentran a la derecha del operador punto y coma (;) que cierra la declaración de un identificador o variable. El comentario de una línea utiliza dos backslash consecutivos, es decir, // y la línea de comentarios; por ejemplo: i=0; //inicializa contador (comentario de una línea).

Otra forma de manejar comentarios con una línea es por medio de los símbolos: backslash seguido de un asterisco /*, posteriormente viene la línea de comentarios, cerrándolo con un asterísco y backslash */. Por ejemplo:

i=i+1;/*incrementa contador (otra forma de comentario de una línea).*/

Cuando el comentario tiene varias líneas de comentarios (uno o más párrafos), el comentario abre con los símbolos *backslash* seguido de un astersco /*, posteriormente viene una línea o párrafos de comentarios, se cierra con asterísco y *backslash* */. Por ejemplo:

/* Comentario de un párrafo que cubren varias líneas con la finalidad de documentar técnicamente en detalle algunas secciones o partes importantes del sketch. */

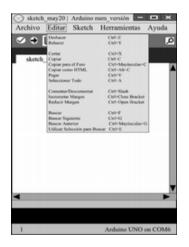


Figura 2.7 Editar.

El menú **Editar** tiene opciones para modificar el tamaño del margen izquierdo en la consola de texto, para mejorar la estética del código del programa, sobre todo para depurar o corregir errores. Otras opciones del modo de edición son: copiar el código del sketch con el formato adecuado para publicarlo en foros de comunidades Arduino o para subirlo a una página Web en formato HTML. La figura 2.7 muestra los comandos que tiene el menú **Editar** y la tabla 2.2 presenta el resumen de combinación de teclas para una fácil activación.

Una herramienta muy importante que tiene el modo de edición es la búsqueda automática de las llaves para abrir { y cerrar } que utilizan las subrutinas, instrucciones de lazo de programación for, while, do-while y en las instrucciones condicionales if, if-else, switch.

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

Fernando Reyes Cortés • Jaime Cid Monjaraz

Particularmente, esta herramienta de búsqueda cobra mayor relevancia si el código del sketch crece y cuando se utilizan instrucciones **if**, **while**, **switch** y **for** anidados, es frecuente perder la llave de cierre o apertura; únicamente colocando el cursor sobre la llave, el modo de edición detecta la llave correspondiente, cuando no la encuentra, para el usuario es fácil detectar la llave faltante.

Tabla 2.2 Comandos del menú Editar.

| Descripción del comando | Combinación de teclas |
|--------------------------------|-----------------------|
| Deshacer | Ctrl+Z |
| Rehacer | Ctrl+Y |
| Cortar | Ctrl+X |
| Copiar | Ctrl+C |
| Copiar para el Foro | Ctrl+Mayúsculas+C |
| Copiar como HTML | Ctrl+Alt+C |
| Pegar | Ctrl+V |
| Selccionar Todo | Ctrl+A |
| Comentar/Descomentar | Ctrl+Slash |
| Incrementar Margen | Ctrl+ Close Bracket |
| Reducir Margen | Ctrl+Open Bracket |
| Buscar | Ctrl+F |
| Buscar Siguiente | Ctrl+G |
| Buscar Anterior | Ctrl+Mayúsculas+G |
| Utilizar Selección para Buscar | Ctrl+E |

Menú Sketch

El menú Sketch (ver figura 2.8) verifica la sintaxis del código fuente del programa sketch de acuerdo con las reglas gramaticales del lenguaje C (Ctrl+R). Cuando hay errores de sintaxis en la zona de mensajes se indican la fuente o causa del error (por debajo de la barra horizontal de deslizamiento).

Alfaomega

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

FERNANDO REYES CORTÉS • JAIME CID MONJARAZ

Una vez depurado este paso y cuando el sketch no tiene errores de programación, entonces se pasa a la fase de compilación para generar el código de máquina del microcontrolador y quedar listo para la descarga del sketch al microcontrolador.

Para abrir la carpeta de programas sketch se oprimen simultáneamente las teclas Ctrl+K, manteniendo ambas teclas, hasta que aparezca la información requerida.

- El icono es una forma más rápida para verificar y compilar el código en lenguaje C y C++ del programa sketch.
- Para añadir un programa fuente sketch se emplea la opción **Agregar Archivo**; el programa aparecerá en una nueva pestaña del ambiente de programación. Las extensiones para cargar un programa son: .c, .cpp y .h, que corresponden a los lenguajes C y C++, así como el código o programas cabecera (header), respectivamente.
- La opción **Importar Librería** permite colocar una librería en la cabecera del programa sketch para realizar una aplicación específica sobre un recurso de la tarjeta; por ejemplo, para manejo de la memoria EEPROM se utiliza: **#include <EEPROM.h>** y para control de motores a pasos se emplea **#include <Stepper.h>**.



Figura 2.8 Menú Sketch.

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

Fernando Reyes Cortés • Jaime Cid Monjaraz

La tabla 2.3 presenta la combinación de teclas para activar los comandos del menú sketch.

Tabla 2.3 Comandos del menú Sketch.

| Descripción del comando | Combinación de teclas |
|---------------------------|-----------------------|
| Verificar/compilar | Ctrl+R |
| Mostrar la carpeta Sketch | Ctrl+K |

Menú Herramientas

El menú **Herramientas** se muestra en la figura 2.9 con las siguientes opciones:

- Formato automático Ctrl+T que, a la vez que proporciona estética al código del sketch, distribuye adecuadamente las tabulaciones entre las llaves de apertura { y cierre } de las instrucciones como if{...else...}, for{...}, así como de subrutinas o funciones.
- Archivar el sketch: respalda la información de un programa en un directorio o una localidad asignada por el usuario.
- Reparar Codificación y Recargar: esta opción recarga de nuevo el sketch, preguntando antes al usuario si descarta todos los cambios o las modificaciones hechos sobre el programa.
- Una vez que se ha descargado el código de máquina del sketch y su ejecución en la tarjeta Arduino, el Monitor serial puede ser activado directamente en el menú de Herramientas o utilizando la combinación de teclas (Ctrl+Mayúsculas+M). El monitor serial es la forma de desplegar la información de parámetros, variables y funciones del sketch, el cual está ejecutándose en la tarjeta Arduino. Esta opción representa una interface de comunicación serial que permite enviar y recibir datos o información entre la tarjeta Arduino y la computadora para su presentación en forma de cadenas de texto; la comunicación serial es bidireccional, es decir, a través del monitor serial también se puede transmitir información hacia la tarjeta. Esta función también se encuentra disponible por medio del icono (mayores detalles pueden ser consultados en la sección 2.4.2).
- Tarjeta: se refiere al modelo de tarjeta electrónica Arduino que el usuario tiene conectada a la computadora por medio del puerto USB. En esta parte el usuario debe seleccionar un modelo específico de tarjeta, ya que de eso depende la configuración interna de parámetros para la compilación. La descarga de código de máquina es realizada en forma automática.
- Puerto serial: es la conexión tipo USB de la computadora con la tarjeta Arduino. Representa el medio para enviar y recibir información de datos de la tarjeta con la computadora.
- **Programador**: permite seleccionar el tipo de hardware para programar una tarjeta sin usar la conexión USB. Normalmente esta opción no se requiere a menos que se quiera grabar el

cargador y arrancador de código bootloader en un nuevo microcontrolador ATMEL, el cual viene sin el cargador de código de programas (sketchs).

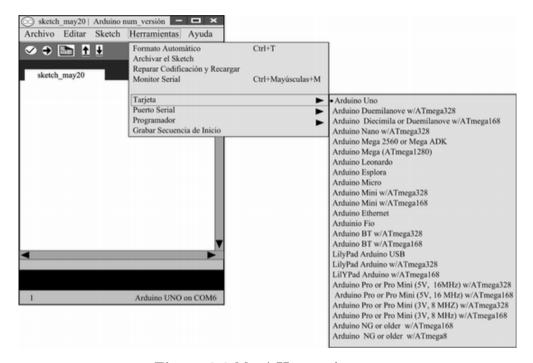


Figura 2.9 Menú Herramientas.

La tabla 2.4 muestra el resumen de comandos del menú **Herramientas** que se activan por combinación simultánea de teclas.

Tabla 2.4 Comandos del menú Herramientas.

| Descripción del comando | Combinación de teclas |
|-------------------------|-----------------------|
| Formato automático | Ctrl+T |
| Monitor serial | Ctrl+Mayúsculas+M |

Menú de Ayuda

Las opciones del menú **Ayuda** representan un enlace directo al sitio Web **www.arduino.cc**, donde se ofrece información y soporte del hardware, software, aspectos técnicos de programación de puertos, memoria y en general aplicaciones en ingeniería de las tarjetas Arduino.

El menú Ayuda se muestra en la figura 2.10 y contiene las siguientes opciones:

- Empezando: es una opción que describe el procedimiento básico de instalación del software de las tarjetas Arduino.
- Entorno: es una guía básica sobre las funciones y características del entorno o ambiente de programación.
- Solución de problemas frecuentes: relacionados con el cargado de programas sketch en las tarjetas Arduino; conflictos con los sistemas operativos Windows, MaC OS X, Linux, así como errores numéricos de compilación con Java, configuración de tarjetas, etc.
- Referencia: es un compendio del lenguaje de programación Arduino (lenguaje C), el cual incluye sintaxis, tipo de datos, instrucciones, operadores aritméticos y lógicos, funciones matemáticas y descripción de funciones especiales para manejo de puertos, memoria, convertidores analógico/digital y digital/analógico, etc. En este apartado, también se encuentran disponibles una variedad de ejemplos didácticos que ilustran la forma de utilizar instrucciones y librerías del sistema Arduino.
- Buscar en la Referencia: descripción específica de funciones o instrucciones de programación Arduino, para lo cual se requiere seleccionar sobre la consola de texto del ambiente de programación el tipo de instrucción o función y combinar las teclas Ctrl+Mayúsculas+F; entonces de manera automática se realiza un enlace al sitio Web: https://www.arduino.cc, donde se proporciona información completa del elemento seleccionado.
- Preguntas frecuentes: enlaza al sitio FAQ (Frequently Asked Questions) del Web: www.arduino.cc, donde se describe la respuesta a un conjunto de preguntas comunes sobre el sistema Arduino.
- Visitar Arduino.cc: es un enlace directo al sitio Web: https://www.arduino.cc.
- Acerca de Arduino: este apartado describe el conjunto de personas que conforman el equipo de diseño y desarrollo del sistema Arduino.

La tabla 2.5 muestra la combinación de teclas del comando de búsqueda que se encuentra dentro del menú **Ayuda**.

Tabla 2.5 Comando del menú Ayuda.

| Descripción del comando | Combinación de teclas |
|-------------------------|-----------------------|
| Buscar en referencia | Ctrl+Mayúscula+F |

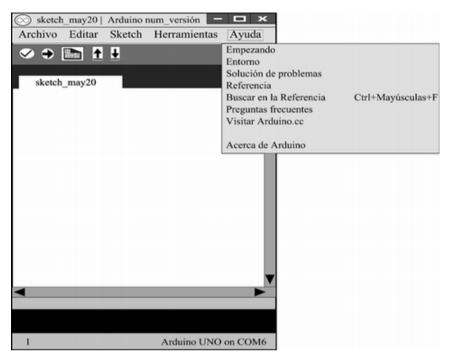


Figura 2.10 Menú Ayuda.

2.4 Puesta a punto



La puesta a punto del sistema Arduino significa comprobar que todos los comandos y herramientas del paquete de cómputo, así como su enlace de comunicación serial con diversos modelos de tarjeta electrónica funcionen correctamente. Es decir, el sistema en este momento se encuentra listo para desarrollar aplicaciones de ingeniería y ciencias exactas. La descripción de este proceso se realizará a través de dos ejemplos ilustrativos didácticos que se encuentran incluidos en el paquete de cómputo: blink y DigitalReadSerial, en tanto que la tarjeta electrónica de trabajo será el modelo Arduino UNO.



2.4.1 Ejemplo blink

El programa de cómputo Arduino contiene varios ejemplos didácticos que ayudan a entender no sólo los recursos que tiene el sistema en forma integral, también su extensión a diversas aplicaciones de ingeniería robótica y mecatrónica; entre los ejemplos más simples se encuentra el programa **blink**, el

cual hace uso de un puerto digital programado como salida para encender o activar por un segundo un LED y apagarlo por otro segundo, ciclo que se repite de manera indefinida.

El sketch **blink** es un programa básico que sirve para encender y apagar por intervalos de un segundo un indicador luminoso LED que se encuentra en la parte superior de la tarjeta electrónica modelo Arduino UNO; tal y como se muestra en la figura 2.11. El tiempo de encendido y apagado puede ser variable, es decir, es definido por el usuario.

Posteriormente se explicará la forma de conectar externamente un LED sobre la terminal de señales disponibles para interface de esta misma tarjeta.

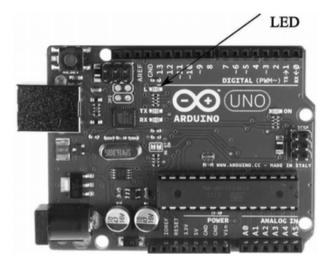


Figura 2.11 Indicador LED del modelo UNO.

A continuación se describen los pasos necesarios para ejecutar el programa ejemplo **blink** en la tarjeta electrónica modelo Arduino UNO.

Procedimiento para correr el sketch blink en el modelo Arduino UNO

El procedimiento inicia al ejecutar el ambiente de programación Arduino (), luego hay que conectar la tarjeta electrónica modelo UNO al puerto USB de la computadora. Ahora, en el menú **Herramientas**, ubique la opción **Tarjeta** y seleccione el modelo Arduino UNO; la figura 2.12 indica la secuencia de pasos para dar de alta a la tarjeta seleccionada.

Verifique en el menú **Herramientas**, opción **Puerto Serial**, que el puerto USB de la computadora esté habilitado con la tarjeta modelo UNO, como se indica en la figura 2.13, la cual señala como un posible ejemplo COM6. Sin embargo, esto dependerá de las características técnicas de su computadora personal.

En el menú **Archivo**, se debe ir a la opción **Ejemplos**, y ahí ubicar **Basics**, seleccionar el sketch **blink**; la figura 2.14 muestra la secuencia de pasos para encontrar el ejemplo indicado. El sketch **blink** es uno de los ejemplos básicos de programación del sistema Arduino, explica en forma simple

cómo programar puertos entrada/salida (I/O). La estructura de programación que debe tener cualquier desarrollo de aplicación se muestra en la figura 2.15; no obstante, mayores detalles sobre el desarrollo de sketchs o programas y sus aplicaciones consultar el capítulo 4 **Lenguaje C**.

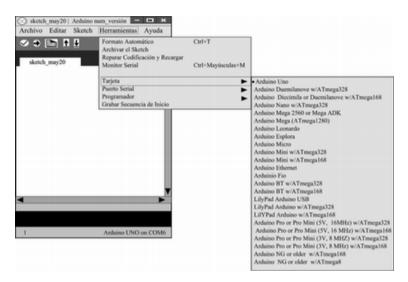


Figura 2.12 Seleccionar el modelo de tarjeta Arduino UNO.

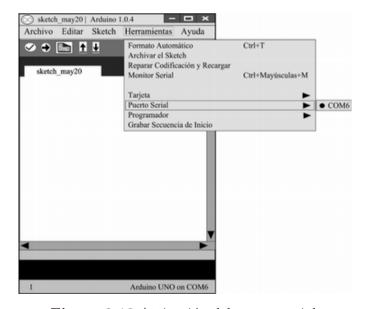


Figura 2.13 Activación del puerto serial.



Figura 2.14 Selección del sketch blink.

La figura 2.15 muestra el área de edición del ambiente de programación Arduino con el código fuente del sketch blink. Este programa activa (enciende) un LED por un segundo y lo apaga o desactiva por otro segundo. Además, no contiene librerías en la parte inicial del programa (cabecera o header). La variable global led es del tipo entero (int) y se emplea para direccionar el pin 13 del puerto digital, en cuyo caso se configura como puerto de salida utilizando la función pinMode(led) en la rutina setup. La rutina loop se ejecuta de manera indefinida hasta que se descargue otro programa o sketch. Esta rutina emplea las funciones digitalWrite para escribir valores alto (High) y bajo (low) en el puerto digital de salida especificado en el pin 13; por medio de la función delay se generan retardos de tiempo por intervalos de un segundo o 1000 mseg.



Figura 2.15 Sketch blink.

Los pasos para compilar, generar código de máquina, descargar y ejecutar el ejemplo blink en la tarjeta Arduino UNO se describen a continuación:

Compilar el programa por medio del comando 父 ; en esta fase se revisa la sintaxis del

Alfaomega

Arduino, Aplicaciones en Robótica y Mecatrónica

Fernando Reyes Cortés • Jaime Cid Monjaraz

sketch **blink** de acuerdo con las reglas gramaticales del lenguaje C, con lo que se genera el correspondiente código de máquina del microcontrolador ATMEGA 328P, el cual representa la plataforma electrónica del modelo UNO.



Descargue el código de máquina del microcontrolador ATMEGA 328P por medio del comando ... Durante la descarga de código, observe cómo parpadean los LEDs de recepción y transmisión Rx y Tx, respectivamente, ubicados en la cara superior de la tarjeta electrónica.

Observe cómo el LED que se encuentra en la cara superior de la tarjeta electrónica UNO enciende y apaga, por intervalos periódicos de un segundo. Este intervalo de tiempo puede ser modificado por el usuario.

De manera opcional, puede conectar en forma externa un LED en las terminales de la tarjeta electrónica UNO; para limitar la corriente que pasa por el LED es conveniente conectar en serie una resistencia de 220 Ω de $\frac{1}{4}$ W. Por ejemplo, la terminal cátodo del LED se conecta a la terminal de tierra (gnd); y una de las terminales de la resistencia conectada al pin 13, como se ilustra en la figura 2.16a, mientras que la figura 2.16b muestra el diagrama esquemático.

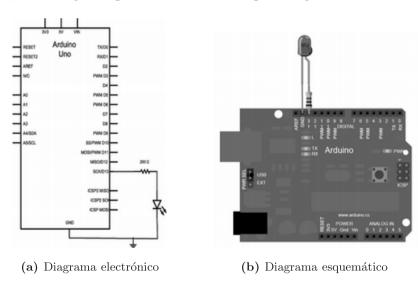


Figura 2.16 Conexiones eléctricas en la tarjeta Arduino UNO del ejemplo blink.

El código fuente del ejemplo **blink** puede ser generalizado para varias aplicaciones (para ver los detalles del código fuente visite el sitio Web del libro correspondiente a este capítulo), por ejemplo:



Encender y apagar luces de habitaciones; en este caso, se emplea como circuito interface un optoacoplador MOC3030 o MOC3040.



Controlar la dirección de puertas o ventanas automáticas.



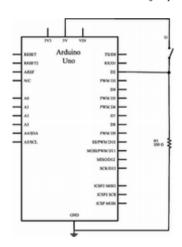
Encender y apagar bombas de agua.



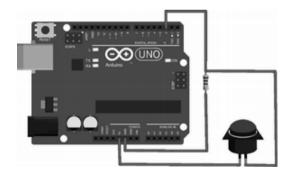
2.4.2 Ejemplo DigitalReadSerial

El segundo ejemplo para comprobar la **puesta a punto** del ambiente de programación y de la tarjeta Arduino es **DigitalReadSerial**, que consiste en leer la información digital de un interruptor y transmitirla a la computadora. El interruptor utilizado es de los denominados permanentemente abierto (*push-button*), el cual está conectado a un puerto digital programado como entrada, que corresponde al pin 2 del conector de señales de la tarjeta Arduino modelo UNO.

Los componentes eléctricos que se necesitan para el ejemplo **DigitalReadSerial** son: una resistencia de 330 Ω de $\frac{1}{4}$ W y un interruptor permanentemente abierto. La figura 2.17a muestra el diagrama electrónico y su correspondiente diagrama esquemático en la figura 2.17b. Normalmente el interruptor tiene un estado lógico 0 (low), pero cuando se presiona entonces cierra el circuito con la señal de 5V, de donde se obtiene el estado lógico alto (high), de esta forma el puerto digital (pin 2) puede leer el estado de voltaje que proporciona el interruptor.







(b) Diagrama esquemático del ejemplo Digital-ReadSerial.

Figura 2.17 Conexiones eléctricas en la tarjeta Arduino UNO del ejemplo DigitalReadSerial.

En el ambiente de programación Arduino, menú **Archivo**, opción **Ejemplos**, habrá que seleccionar **Basics** y ubicar **DigitalReadSerial**; la figura 2.18 muestra la secuencia de pasos para descargar el

sketch a la zona de edición del ambiente de programación, mientras que en la figura 2.19 se ilustra el código fuente del sketch **DigitalReadSerial**.



Figura 2.18 Selección del sketch DigitalReadSerial.

El ejemplo **DigitalReadSerial** lee el estado digital del pin 2 configurado como puerto digital de entrada, cuyo resultado se envía al monitor serial. Un interruptor permanentemente abierto se conecta al pin 2 (variable tipo entera **pushButton**); la rutina de configuración **setup** contiene la función **Serial.begin(9600)** para programar la velocidad de transmisión serial (9600 bits por segundo) y la configuración del puerto digital 2 como puerto de entrada, lo que ocurre utilizando la función **pinMode(pushButton, INPUT)**.

El lazo principal de programación loop lee el estado digital del puerto digital de entrada por medio de la función digitalRead(pushButton), cuyo resultado se almacena en la variable de tipo entero buttonState, para transmitir al monitor serial a través de la función Serial.println(buttonState). La ventana del monitor serial se activa con el icono para desplegar la información que contiene la variable buttonState (véase la figura 2.20b). Finalmente se utiliza la función delay(1) para generar un milisegundo de retardo entre envíos consecutivos de transferencia serial. El lazo de programación loop repite de manera indefinida este conjunto de instrucciones hasta cargar un nuevo sketch, producir un reset o desconectar la alimentación de la tarjeta (fuente externa o por falta de la fuente, entonces el cable de comunicación USB).



Figura 2.19 Código fuente del sketch DigitalReadSerial.

Los pasos para compilar, generar código de máquina, descargar y ejecutar el ejemplo **DigitalReadSerial** en la tarjeta electrónica modelo UNO se describen a continuación:

- Compilar el programa por medio del comando \bigcirc ; en esta fase se revisa la sintaxis del sketch **DigitalReadSerial** de acuerdo con las reglas del lenguaje C, lo que genera el correspondiente código de máquina del microcontrolador ATMEGA 328P para el modelo UNO.
- Descargar el código de máquina hacia la tarjeta electrónica modelo UNO por medio del comando . Cuando se compila y descarga el código de máquina del sketch **DigitalRead-Serial**, se despliega en la parte inferior de la ventana la leyenda "carga terminada", que también indica el tamaño del sketch como se muestra en la figura 2.20a.



Aplicaciones de blink

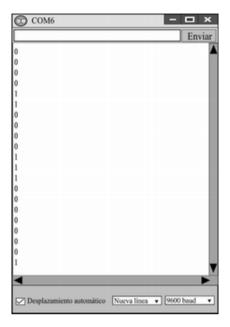
En el sitio Web del libro, el lector puede consultar aplicaciones del sketch **blink** como las que a continuación se describen: apagado y encendido de luces, cambio de giro de motores, accionamiento de bombas de agua, etc.



Aplicaciones de DigitalReadSerial

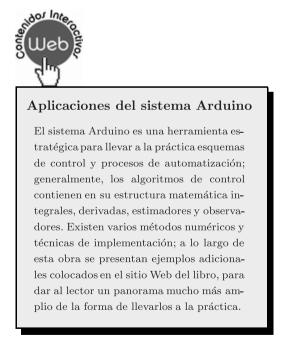
El sketch **DigitalReadSerial** se puede generalizar para cubrir varias aplicaciones como: interruptores *limit switch*, detectores de puerta abierta, control de bandas industriales. Visite sitio Web del libro, para consultar otros ejemplos demostrativos.





(a) Descarga del sketch DigitalRead-Serial. (b) Monitor serial.

Figura 2.20 Descarga y monitor serial del ejemplo DigitalReadSerial.





2.5 Resumen

En N este capítulo se han presentado el proceso de instalación y puesta a punto del sistema Arduino, el cual consta de dos componentes principales:

- Paquete de cómputo, que es el entorno o ambiente de programación (IDE) orientado al desarrollo de aplicaciones de ingeniería robótica y mecatrónica, automatización de procesos físicos y en ciencias exactas.
- Tarjeta electrónica, que depende del tipo de microcontrolador ATMEL y de sus componentes periféricos integrados definen el tipo de modelo.

El sistema Arduino maneja la filosofía de software libre y hardware en arquitectura abierta:

- Software libre (free software): significa que se tiene libertad de usarlo para cualquier propósito (licencia libre) y en cualquier sistema operativo; también, para adaptarlo a necesidades particulares y específicas. Aquí hay que tomar en cuenta que tener acceso al código fuente es un requisito previo; lo anterior le da la libertad para distribuir copias, así como incorporar mejoras al programa y hacerlas públicas para beneficio de la sociedad.
- Hardware en arquitectura abierta (open-source hardware): permite estudiarlo para entender su funcionamiento, modificarlo y compartir dichos cambios; asimismo, publicar en foros sociales los diagramas electrónicos y esquemáticos, es decir, tiene el objetivo de facilitar la tecnología moderna al público en general; no hacerlo de manera pasiva o para consumo, más bien en forma activa involucrando al usuario para que obtenga mayor valor por la tecnología desde un punto de vista ético y por ende mejore su educación y cultura.

Software libre y hardware en arquitectura abierta son conceptos de la tecnología moderna que ayudan a la comunidad para mejorar su estilo y calidad de vida, lo que permite realizar aplicaciones específicas $(ad\ hoc)$ en ingeniería robótica y mecatrónica acordes a los requerimientos y las necesidades del proceso a controlar o automatizar.

El potencial del sistema Arduino puede ser mejorado cuando se emplea con otro tipo de paquetes, por ejemplo con MATLAB; ver el capítulo 8 Arduino con MATLAB donde el lector puede encontrar detalles específicos de cómo realizar adquisición de señales y su representación gráfica.

2.6 Referencias selectas



E XISTE una enorme cantidad de referencias bibliográficas que describen el sistema Arduino. De manera particular, se recomienda al lector los siguientes libros de consulta:

- Brian W. Evans. "Arduino programming notebook". Creative Commons. Second edition. September 2008.
- Jonathan Oxer and Hugh Blemings "Practical Arduino: Cool projects for open source hardware". Appres, 2009.
- Maik Schmidt. "Arduino a quick-start guide". The Pragmatic Bookshelf. 2011.
- J. M. Hughes "Real world instrumentation with python". Published by O'Reilly Media, Inc. First edition, 2011.
- John David Warren, Josh Adams and Harald Molle. "Arduino Robotics". Apress, 2011.
- Emily Gertz and Patrick Di Justo. "Environmental monitoring with Arduino". Published by O'Reilly Media, Inc. 2012.
- http://www.arduino.cc
- Web http://www.wiring.org.co
- http://www.arduino.cc/en/Booklet/HomePage
- web http://cslibrary.stanford.edu/101/



2.7 Problemas propuestos

Esta sección se destinó a presentar al lector un conjunto de problemas para reflexionar y al mismo tiempo poner a prueba los conocimientos adquiridos sobre la instalación y puesta en marcha del sistema Arduino.

- 2.7.1 Describa cuáles son los pasos necesarios para instalar el paquete de cómputo del sistema Arduino.
- 2.7.2 Describa el procedimiento de instalación para el sistema operativo Windows.
- 2.7.3 Describa el proceso de instalación para el sistema operativo MacOS X.
- 2.7.4 ¿Cómo se habilitan los controladores o drivers del puerto USB y su reconocimiento para los modelos de tarjeta Arduino?
- 2.7.5 ¿Cuáles son las funciones importantes que tiene integradas el ambiente de programación Arduino?
- 2.7.6 Para el ambiente de programación Arduino, describa la secuencia requerida para compilar y descargar el código de máquina de un sketch hacia un modelo determinado de tarjeta Arduino.
- 2.7.7 ¿Cómo puede visualizar el valor numérico de una variable de un programa o sketch en el ambiente de programación Arduino?
- 2.7.8 ¿Cómo puede proceder para el caso de las versiones del sistema operativo Windows 8.0 y 8.1 que no permitan la instalación de los drivers de las tarjetas Arduino?
- 2.7.9 Modifique el tiempo de encendido y apagado del diodo LED en el programa **blink**, de tal forma que el tiempo de encendido sea 5 segundos y el tiempo de apagado 4 segundos.
- 2.7.10 En relación con la figura 2.20a, la descarga de código del ejemplo **DigitalReadSerial** es de 2,922 bytes de un máximo 32,256 bytes.
 - a) Proporcione una explicación técnica sobre la diferencia de tamaño del código de máquina del sketch **DigitalReadSerial** 2,922 bytes en relación con el tamaño de 32,256 bytes.
 - b) ¿Por qué el proceso de compilación únicamente genera una transferencia de 2,922 bytes?
 - c) ¿Cuál es el significado de un tamaño máximo de 32,256 bytes?
- 2.7.11 Explique el concepto de software libre.
- 2.7.12 ¿Qué significa open-source hardware?

Plataforma electrónica



Capítulo Web

- 3.1 Introducción
- 3.2 Arquitectura AVR
- 3.3 Plataforma electrónica Arduino
- 3.4 Modelos de tarjetas Arduino
- 3.5 Resumen
- 3.6 Referencias selectas
- 3.7 Problemas propuestos

Competencias

Presentar los conceptos fundamentales de la arquitectura AVR de los microcontroladores ATMEL; plataforma electrónica Arduino y modelos de tarjetas, así como sus potenciales aplicaciones.

Desarrollar habilidades en:

- Descripción de la arquitectura AVR.
- Tabla y manejo de interrupciones.
- Organización de la memoria para programas y datos.
- Tipos de periféricos disponibles en las tarjetas Arduino.
- Formas de comunicación serial.
- Gestor de arranque y cargador de código de máquina.
- Aspectos funcionales del modelo Arduino UNO.
- Descripción técnica de las señales para interface electrónica de la tarjeta Arduino UNO.
- Tipos de tarjetas de la plataforma electrónica Arduino y sus potenciales aplicaciones.

```
int i, j; float C[3][3], A[3][3], B[3][3]; void loop() {  for(i=0;\,i{<}3;\,i{+}{+})\{ \\ for(j=0;\,j{<}3;\,j{+}{+})\{ \\ C[i][j]{=}A[i][j]{+}B[i][j]; \\ \} \\ \} \\ \}
```

- 4.1 Introducción
- 4.2 Empezando a programar en C
- 4.3 Variables
- 4.4 Operadores
- 4.5 Arreglos
- 4.6 Funciones
- 4.7 Instrucciones de programación
- 4.8 Resumen
- 4.9 Referencias selectas
- 4.10 Problemas propuestos

Competencias

Presenta los fundamentos esenciales del lenguaje C a través de programas didácticos y soporte pedagógico con ejemplos documentados que ilustran las características esenciales de sintaxis y gramática (reglas formales de programación) de operadores (a nivel bytes y bits), identificadores, tipos de datos, arreglos, sentencias, instrucciones, funciones y estilo de programación para proporcionar al lector de los elementos necesarios que le permitan dominar el lenguaje de programación C y la implementación práctica en las tarjetas Arduino con aplicaciones en ingeniería y ciencias exactas.

Desarrollar habilidades en:

- Dominio de las reglas de sintaxis para la escritura de programas en lenguaje C.
- Que el lector sea capaz de programar aplicaciones útiles que se ejecuten en las tarjetas Arduino para robótica, mecatrónica y ciencias exactas.
- Manejo de operadores aritméticos, lógicos y relacionales.
- Manejo de tipos de datos y el uso adecuado de modificadores.
- Conocimiento pleno de las instrucciones de programación.
- Diseño de código eficiente de funciones o subrutinas.

4.1 Introducción 53

4.1 Introducción



E lenguaje C fue desarrollado por Dennis Ritchie usando como sistema operativo DEC PDP-11 y también fue pionero de Unix junto con Ken Thompson. Hoy en día, existen compiladores de C para una variedad muy amplia de microcontroladores, computadoras y sistemas operativos; el lenguaje C se ha convertido en la base de todas las aplicaciones prácticas de ingeniería y ciencias exactas.

C permite la manipulación a nivel bits y bytes, palabras (ancho o número de bits del bus de datos de un microcontrolador), punteros, arreglos, operaciones aritméticas sobre los valores que puede tener una variable (definida como entera o real), direcciones y acceso a memoria, manejo de puertos de entrada/salida. El lenguaje de programación C es ideal para implementar aplicaciones de física, matemáticas, control, automatización de procesos en mecatrónica y robótica.

El lenguaje C es programación estructurada a bloques, permite declarar procedimientos o funciones dentro de otras funciones, también lo realiza en forma recursiva, es decir se puede ejecutar una función dentro de sí misma. De esta forma, se extienden los conceptos de variables globales y locales para controlar la visibilidad de éstas mediante el uso de gramática o sintaxis propia de este lenguaje.



C es considerado un lenguaje de computadora de nivel medio debido a que presenta elementos de programación del lenguaje de alto nivel (Ada, Modula-2, Pascal, Cobol, Fortran, Basic, etc.) con el funcionamiento del lenguaje ensamblador y macroensamblador.



Esta característica de lenguaje de nivel medio, le permite adaptabilidad y potencia de programación, ya que permite realizar operaciones y manipulación sobre bits, bytes, direcciones de memoria, puertos y periféricos de interfaces electrónicas, así como uso de diferentes tipos de datos como enteros, números reales, doble precisión, arreglos, uniones y estructuras de datos.



También posee la flexibilidad de las instrucciones de programación de los lenguajes de alto nivel.



El lenguaje C permite reprogramar los recursos de una computadora y adaptarlos a requerimientos de control y automatización, de tal forma que pueda funcionar como un sistema empotrado.



C presenta una importante característica que se denomina **portabilidad**, lo cual significa que es posible ejecutar el código fuente de cualquier computadora, por ejemplo un programa en lenguaje C puede ser compilado para los sistemas operativos Linux, Mac o Windows, así como en diferentes tipos de microcontroladores.



Portabilidad es un aspecto característico clave, que resulta estratégico para programar algoritmos de procesamiento de señales, esquemas de control, métodos numéricos en las tarjetas electrónicas del sistema Arduino.



El lenguaje C es la columna vertebral del lenguaje C++, el cual es portable en cualquier sistema operativo, así como a la plataforma Arduino.

Una cualidad del lenguaje C que se convierte en un rasgo distintivo es el comportamiento de datos y código, esto significa que es la capacidad de este lenguaje para seccionar y esconder del resto del programa información innecesaria y al mismo tiempo mantener instrucciones de programación específica para llevar a cabo correctamente la tarea programada. Por ejemplo, las funciones o subrutinas manejan variables locales, esto significa que son variables temporales que sólo se activan dentro de la función y que no provocan efectos secundarios en otras partes del programa. De ahí que lo importante de una función en C es saber qué es lo que hace, y no cómo lo hace.

Este capítulo presenta los fundamentos del lenguaje C, gramática y sintaxis de identificadores, tipos de datos que definen a las variables, constantes, arreglos, funciones, operadores aritméticos, lógicos, relacionales, instrucciones de programación.

La exposición de cada tema se realiza a través de documentación y con el apoyo de diagramas de flujo, así como una serie de ejemplos didácticos, cuidadosamente seleccionados con la finalidad de que el usuario pueda entender los aspectos cualitativos de la programación de algoritmos en lenguaje C y ejecutarlos en las tarjetas electrónicas Arduino sin la necesidad de realizar conexiones.

4.2 Empezando a programar en C



E lenguaje C tiene elementos de programación y reglas gramaticales para desarrollar programas; la gramática se refiere al conjunto de reglas o sintaxis para escribir correctamente los elementos de programación. El lenguaje de programación C utiliza identificadores para declarar o asignar un nombre a las variables que pueden ser de diversos tipos de datos tales como enteros de un byte hasta 4 bytes, números reales (flotantes y con doble precisión), operadores de tipo aritmético y lógico que manipulan el valor numérico de las variables (a nivel de bits o bytes), arreglos, apuntadores, instrucciones lógicas y control de lazos de programas, funciones o subrutinas que realizan procesamiento o cálculos específicos.



Estándar ANSI

El lenguaje de programación C se basa en el estándar ANSI (American National Standards Institute) publicado por el Instituto Nacional Estadounidense de Estándares, el cual define que los programas en C deben satisfacer características específicas para garantizar portabilidad en diferentes plataformas y sistemas operativos.

Mediante archivos de texto (ASCII) se realiza la edición de los programas, la estructura del lenguaje C para el sistema Arduino consiste en secciones específicas para librerías, variables globales, funciones o subrutinas, la función de configuración void setup() de puertos, periféricos, comunicación USB y la función principal denominada void loop() la cual contiene un conjunto de sentencias e instrucciones que realizan un procedimiento específico, dada una aplicación.

La parte inicial de un programa en C se utiliza para incluir librerías (son funciones que ya se encuentran compiladas) mediante la directiva #include <archivo_libreria.h>, la extensión de las librerías es .h (punto h), significa header o cabecera. Las librerías se incluyen cuando la aplicación lo requiere, es decir son opcionales; por ejemplo, cuando se requiere trabajar con funciones matemáticas como e^x entonces se utiliza #include <math.h>. Se pueden incluir tantas librerías como sean necesarias (mayores detalles sobre librerías ver el capítulo 6 Librerías), continúa opcionalmente con la declaración de las variables globales del programa, funciones o subrutinas como la función de configuración void setup() para establecer la forma de trabajo de los puertos entrada/salida, velocidad de comunicación serial y el código principal que identifica a la plataforma Arduino void loop() $\{...\}$, la cual contiene el algoritmo de la aplicación a realizar.

El código ejemplo 4.1 muestra la estructura básica que debe mantener un programa o sketch

para el sistema Arduino; observe que el sketch inicia con librerías en el header o cabecera del sketch que corresponde a las primeras líneas del programa. Para incluir una librería se utiliza #include <archivo.h> o también pueden ir entre comillas #include "archivo.h"; las librerías no contienen código fuente en C, sólo la sintaxis que indica la forma correcta de escribir la función, es decir el tipo de datos que utilizan sus argumentos y cómo lo retornan, inclusive si no retorna datos (tipo void). La extensión .h indica que es un archivo que se incluye en el header del programa o sketch.

Observe que en las primeras líneas del código ejemplo 4.1 se encuentran las librerías para funciones matemáticas, generales, estándar, Ethernet motores etc.; Algunas librerías como **math.h**, **stdlib.h** y **stdio.h** se pueden usar sin la necesidad de tener conectado a la tarjeta Arduino algún módulo, dispositivo o motor, inclusive si no se usan las funciones de esas librerías, no generan errores, ni problemas de compilación, pero ciertas librería como **Servo.h** y **Ethernet.h** requieren conexión de los módulos de servomotores y Ethernet, respectivamente (véase capítulo 6 **Librerías**, sección 6.2 **Librerías Arduino**, página 130).

Después de incluir las librerías, se declaran las variables globales que pueden ser de cualquier tipo válido del lenguaje C (ver sección **Variables**, página 61) como pueden ser del tipo entero, flotante, etc. Las variables globales son visibles para cualquier función y también para la rutina principal de ejecución **void loop()**.

Continuando con la estructura del sketch, siguen las funciones, también conocidas como rutinas, subrutinas o procedimientos. Cada función se debe indicar el tipo de dato que retorna (ver sección 4.6 Funciones, página 87) e inclusive, también puede suceder que no retornen dato alguno, en este caso corresponde al tipo void el cual sólo es específico para funciones y no se aplica a variables. Algunas funciones utilizan argumentos de entrada como eleva_al_cubo(float x) y retorna un tipo de dato flotante, pero también hay funciones que no tienen argumentos de entrada y no retornan datos, este es el caso de la función del sistema Arduino void setup(), la cual configura la velocidad de transmisión/recepción de datos seriales entre la tarjeta Arduino y la computadora, así como la forma de operación de los puertos y periféricos electrónicos de las tarjetas Arduino (ver capítulo 6 Librerías, sección 6.3 Funciones Arduino, página 138).

En el interior de las funciones se utilizan variables de tipo local, es decir sólo existen para propósitos de dicha función y fuera de ella nadie las puede utilizar; este es un potencial del lenguaje C, que hace que aun coincidiendo en nombre con otras variables locales definidas con otros tipos de datos en diferentes funciones, no generen problemas o colapsos del programa.

Observe que en el cuadro de código 4.1 en la sección de variables globales, se declararon \mathbf{y} , \mathbf{x} como variables de tipo flotante, las cuales coinciden en nombre con la variable local de la función

eleva_al_cubo(float \mathbf{x}) y con su argumento de entrada, respectivamente. Sin embargo, físicamente ocupan lugares de memoria diferentes y estrictamente no interfieren las variables globales; dentro de la función eleva_al_cubo(float \mathbf{x}) tienen preferencia la variable local \mathbf{y} y el argumento de entrada \mathbf{x} sobre las variables globales.

La función principal que realiza la ejecución del sketch se llama loop(), es una función del tipo void y contiene un conjunto de instrucciones y sentencias del usuario que hacen ejecutar la implementación en la tarjeta Arduino. Note que dentro de loop() se declaran las variables f, w, que se utiliza en la función trigonométrica sin(...) y la función eleva_al_cubo(float x). Además, también se emplean funciones del sistema Arduino como: Serial.println(...), las cuales sirven para enviar letreros, datos al monitor serial del ambiente de programación Arduino; mayores detalles sobre esta función se encuentran en el capítulo 6 Librerías y funciones Arduino, página 138.

La sintaxis del lenguaje C incluye un conjunto de operadores para delimitar instrucciones y sentencias como parte del código de programación que a continuación se explican.

○ Código ejemplo 4.1

```
Estructura básica de un sketch
//Header o cabecera para librerías.
#include <math.h>//funciones matemáticas.
#include "stdlib.h" //funciones generales del lenguaje C.
#include <stdio.h>//funciones estándar de entrada/salida.
#include "Servo.h" //manejo de servomotor.
/*Declaración de variables globales, visibles para todo el sketch.*/
int i=0,j,k, led=13;//variables globales de tipo entero.
float x,y=0.333;//variables globales del tipo flotante (números reales).
/*Funciones o subrutinas, emplean {...} y paréntesis ().*/
void setup() {//llave que indica el inicio de la rutina de configuración
    Serial.begin(9600);//9600 bits por segundo.
    pinMode(led, OUTPUT);//configura el pin 13 como puerto de salida.
}//llave que cierra el código de la función setup().
float eleva_al_cubo(float x)\{//\text{llave que indica inicio de la función } x^3.
    float y;//variable local, sólo visible para esta función.
    y=x*x*x;//y=x^3
    return y;//retorna resultado de tipo flotante.
\frac{1}{2}//llave que indica la terminación de la función x^3.
void loop(){//llave que indica el inicio de la función principal.
    float f=3.1416, w; //variables locales sólo visibles dentro de loop().
    w=sin(0.8*f+y);//cálculo con frecuencia y fase.
    x=eleva\_al\_cubo(w);//eleva al cubo x = w^3.
    Serial.println("Valor al cubo=" );//letrero en el monitor serial.
    Serial.println(x,4);//envía resultado con 4 fracciones al monitor serial.
}//llave de cierre que indica la finalización de la función loop(...).
```



4.2.1 Operadores básicos del lenguaje C

La estructura básica del lenguaje C utiliza un conjunto de operadores con funciones bien específicas tales como: llaves {...} para delimitar bloques de código, paréntesis (...) para enmarcar los argumentos de entrada de una función o instrucción de programación, el uso de la coma para separar variables o argumentos, punto y coma que indica el término de una sentencia o declaración, comentarios para documentar el programa, etc.

Bloques de comentarios multilínea /*...*/

Los bloques de comentarios o comentarios multilínea, son áreas de texto ignoradas por el compilador del programa y se usan para documentar o describir aspectos técnicos del código. Un comentario multilínea empieza con los símbolos /* y terminan con */ pueden abarcar varias líneas de comentarios. Dentro de un programa o **sketch** comentarios son ignorados y no ocupan espacio en la memoria de la tarjeta Arduino. También se utilizan para deshabilitar bloques de código y se pueden activar de nuevo eliminado /*...*/. En el código ejemplo 4.1 se puede observar el uso de comentarios multilínea.

Comentarios de línea //

Comentarios de una sola línea empiezan con doble slash // y terminan insertando un **enter** (\leftarrow retorno de carro) para generar la siguiente línea de código. A diferencia del comentario multilínea /*...*/, los comentarios de una sola línea no requieren de un símbolo especial para finalizarlo, o en tal caso finalizan con el cambio de línea (oprimiendo **enter** \leftarrow).

Es común usar comentarios de línea después de declaraciones de variables o instrucciones para proporcionar más información sobre qué lleva la declaración o proporcionar un recordatorio al usuario. Los comentarios de una línea y multilínea son ignorados por el compilador del programa y por lo tanto no ocupan espacio en memoria. En el código ejemplo 4.1 se emplean comentarios de una sola línea.

Llaves $\{....\}$

Las llaves definen el comienzo y el final de bloques de función y bloques de declaraciones como las funciones setup(), loop() e instrucciones de programación como for, if. Se utiliza una llave inicial { (ver el código ejemplo 4.1) que indica el inicio de una instrucción de programación o función, posteriormente continúa la declaración de variables, instrucciones, sentencias, finalizando con la llave de cierre }, es decir, las llaves deben estar balanceadas; en otras palabras, para una llave de

apertura { debe seguirle una llave de cierre }. Si las llaves no están balanceadas, significa que una llave de apertura { no tiene su correspondiente llave de cierre } (o viceversa), entonces provoca errores de sintaxis gramatical del lenguaje C que se detectan en el proceso de compilación. El entorno Arduino incluye una práctica característica para probar el balance de llaves; sólo seleccione una llave y su correspondiente llave de cierre o apertura, según corresponda aparecerá resaltada, de no ser así indicará que falta dicha llave.

Operador punto y coma;

Un punto y coma es un elemento de sintaxis en el lenguaje C, debe usarse al final de cada declaración y sirve para separar los elementos del programa. Observe en el código ejemplo 4.1 el uso de punto y coma después de cada sentencia. Por otro lado, olvidar un punto y coma al final de una declaración producirá un error de compilación. También se usa como operador para separar los elementos de la instrucción for (;;) {...} (ver la subsección 4.7.3 for, página 106).

Operador coma,

El operador coma sirve para separar variables o argumentos de entrada de una función o instrucción de programación. Por ejemplo, cuando se declara una lista de variables, se debe usar coma, para separar adecuadamente cada una de las variables:

```
int i=0, j=100, k;//variables de tipo entero. \hookleftarrow float x, y=10,2, z=2.345;//variables de tipo flotante. \hookleftarrow x=controlPD_robot(y, z, j);/*en esta función los argumentos de entrada están separados por comas.*/\hookleftarrow
```

Observe que se permite inicializar variables en la misma etapa de declaración; note el uso del operador punto y coma; para indicar la finalización de la declaración de variables, posteriormente se insertan comentarios de línea asignados al programa para documentar información, el uso de \leftarrow para iniciar una nueva línea de código.

El operador coma, también se utiliza dentro de la instrucción **for** y forma parte de su sintaxis (ver la subsección 4.7.3 **for**, página 106).

& Ejemplo 4.1

Escribir un programa o sketch en lenguaje de programación C que despliegue la frase "Hola mundo" en el monitor serial del ambiente de programación Arduino.

Solución

El cuadro de código Arduino 4.1 contiene el sketch cap4_hola con la programación básica

en lenguaje C para desplegar el letrero "Hola mundo" en el monitor serial del ambiente de programación Arduino.

El sketch inicia en la línea 2 con la declaración de la función void setup() y la llave de apertura {, esta función configura la comunicación serial a 9600 Baudios entre la tarjeta Arduino y la computadora donde radica el ambiente de programación (ver línea 3). La llave de cierre de la función setup() se ubica en la línea 3 indicando el término de esta función.

La función o subrutina principal loop() inicia en la línea 7, observe la llave de apertura {, el código empieza en la línea siguiente; el letrero "Hola mundo.", representa argumento de entrada a la función Serial.println(...), la cual envía como cadena de caracteres o mensaje de texto por comunicación serie USB al monitor serial Arduino y la función delay(3000) genera una pausa de 3 segundos con la finalidad de que el usuario puede visualizar correctamente el letrero (mayor información sobre estas funciones ver capítulo 6 Librerías, sección Funciones Arduino, página 138).

Después de 3 segundos, el sketch cap4_hola se ejecuta una vez más en la tarjeta Arduino el código contenido entre las llaves {...} de la función loop(), este proceso se realiza de manera indefinida hasta que se descargue otro sketch, se retire la fuente de alimentación externa o USB.

```
    Código Arduino 4.1: sketch cap4_hola

  Arduino. Aplicaciones en Robótica y Mecatrónica.
                                                       Disponible en Web
  Capítulo 4 Lenguaje C.
  Fernando Reyes Cortés y Jaime Cid Monjaraz.
  Alfaomega Grupo Editor: "Te acerca al conocimiento".
   Sketch cap4_hola.ino
1 //Rutina de configuración de la tarjeta Arduino.
2 void setup() { //configuración de comunicación serial USB.
       Serial.begin(9600);//9600 bits por segundo.
4 }
6 //Inicia rutina principal.
7 void loop() {//llave que abre indicando inicio de código de la función loop().
       Serial.println("Hola mundo.");//se envía letrero al monitor serial.
       delay(3000);//pausa por 3 segundos y se repite la ejecución del sketch.
10 }
```

4.3 Variables 61



4.2.2 ¿Cómo ejecutar programas o sketchs?

El procedimiento para ejecutar programas o sketchs en lenguaje C sobre una tarjeta del sistema Arduino se encuentra indicado en la sección 2.4 del capítulo 2 Instalación y puesta a punto del sistema Arduino, página 37. No obstante, para comodidad al lector, dicho procedimiento se describe a continuación. Considere el sketch cap4_hola del cuadro de código Arduino, cuyo código fuente se encuentra disponible en el sitio Web de esta obra (uso):

- En el menú **Herramientas** seleccionar el modelo de tarjeta, puerto USB y configurar velocidad de comunicación serial USB en 9600 Baudios.
- Compilar el sketch mediante el icono 🕢.
- Descargar el código de máquina del sketch a la tarjeta Arduino usando 👈.
- Desplegar resultados con el monitor serial (activar con 🔑).
- Importante: para exhibir adecuadamente los resultados del sketch, maximice la ventana del monitor serial y active la opción desplazamiento automático.

4.3 Variables



E lenguaje C define identificadores como nombres que se utilizan en la declaración de variables, constantes, funciones, etiquetas y otros elementos de programación. El primer carácter de un identificador debe ser una letra o guion bajo (underscore o subrayado), posteriormente pueden seguir combinación de números enteros positivos utilizando símbolos del 0 al 9 y combinaciones de éstos, letras del alfabeto castellano minúsculas (a-z) y mayúsculas (A-Z), con excepción de la letra $\tilde{\mathbf{n}}$ y $\tilde{\mathbf{N}}$. La tabla 4.1 muestra ejemplos de identificadores con sintaxis correcta e incorrecta.

La longitud de un identificador puede variar entre uno y varios caracteres; sin embargo, de acuerdo con las normas ANSI establece que los seis primeros caracteres del nombre deben ser significativos si dicho identificador será externo (variables globales y funciones), es decir que se utilizará por varios archivos o involucra un proceso de compilación con diversas librerías. Por otro lado, si el identificador se utiliza dentro de un archivo o como parte interna de una función (variables locales), entonces se requiere que los 31 primeros caracteres sean significativos.

Una variable no sólo representa un identificador o nombre, también significa una forma de almacenar en memoria un valor numérico para usarse después por el programa. Como su nombre

lo indica, las variables son localidades de memoria que registran información que pueden cambiarse continuamente durante la ejecución del sketch, al contrario de lo que sucede con las constantes, cuyo valor numérico nunca cambia en la memoria asignada.

Tabla 4.1 Ejemplos de identificadores.

| Correcto | Incorrecto | Descripción |
|------------|------------|--|
| robot9 | 9robot | Los identificadores no puede iniciar con un número, puede ser confundidos con constantes y generan error de sintaxis. |
| sensor_23 | #sensor_23 | Hay ciertos símbolos reservados para el lenguaje C como # se utiliza en la cabecera (header) del programa para incluir archivos. |
| puerto_uno | puerto.uno | Existen estructuras de datos en lenguaje C con campos internos, su acceso y manipulación es por medio del operador punto. |
| Sin | sin | No se pueden utilizar identificadores con palabras claves o reservadas, en este caso \sin pertenece a la función matemática seno: $\sin(x)$. Particularmente, en el ambiente de programación Arduino las palabras claves aparecen en color naranja. |
| _99 | -99 | En general, los caracteres alfanuméricos @, !, #, \$, +, -, *, /, =, %, (,), [,], i, !, \dot{i} , ?, { y } se emplean como operadores con funciones específicas. |

Para declarar una variable se requiere especificar el tipo de dato que va a almacenar, como por ejemplo: int, long, float, etc. Opcionalmente, en la misma declaración se le puede asignar un valor numérico, esto sólo necesita hacerse una vez en un programa, pero hay que tomar en cuenta que el valor puede cambiarse en cualquier momento usando operaciones aritméticas, operadores o asignaciones numéricas. Todas las variables tienen que ser declaradas antes de que puedan ser usadas; la declaración puede ser en cualquier parte del programa o sketch; el lugar donde ocurra, determina que partes del programa pueden usar a esas variables.

4.3 Variables 63

Ejemplos de declaración de variables son:

int i=0, j=20,k =35; //variables enteras con de dos bytes con valores iniciales.

byte b0,b1,b2://variables enteras de 8 bits.

float x=0.033, y=1.45, z=23.678;//variables de tipo flotante.

long Y=-123456, F=4567, u=23456;//tipo entero con 4 bytes.

short n1=9, j98=123;//variables de tipo entero de un byte.

word n2=234, m9=129;//variables de tipo entero de 2 bytes.

char cadena[6] = "robot" ;//cadena de caracteres con doble comillas.

char c='c', h=130, v='n';//un simple carácter usa una comilla.

boolean bit0=0, bit1=1, bit2=2;

int puerto[] = 0, 3, 5, 7, 9;

double x, y, z=3.1416;

Una vez que una variable ha sido inicializada o asignado un valor numérico, entonces puede ser usado para su manipulación y procesamiento por alguna función, instrucciones de lazo o control de programa, sentencias de tipo condicional, operadores, operaciones aritméticas, etc.



Variables

En el lenguaje de programación C los identificadores de funciones o variables con letras minúsculas y mayúsculas se tratan de manera diferente, por ejemplo la variable **robot** es diferente a las variables con nombre **Robot** y **ROBOT**. Un identificador no puede iniciar con un número, el nombre de la variable **9robot** es incorrecto; sin embargo, son correctos los siguientes nombres de variables: **robot9**, **r9obot**, **_9robot**, **r_9obot**, etc. El primer carácter de los nombres de identificadores no pueden iniciar con los siguientes símbolos: #, \$, +, -, *, /, =, %, (,), [,], i, !, ;, :, {,}, ., ?, ', \, '', coma y punto.



Palabras reservadas

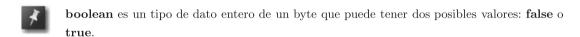
El lenguaje C incluye un conjunto de palabras claves o nombres reservados que no pueden utilizarse como identificadores de variables, por ejemplo no deben tener el nombre de una función escrita o de una librería, tales son los casos de sin, cos, tanh, atan, sqrt, float, long, int, char, void, short, etc.

En el ambiente de programación Arduino las palabras claves aparecen en color naranja, indicando al usuario que son reservadas y por lo tanto, no pueden utilizarse en la declaración de ningún identificador.



4.3.1 Tipos de datos

Las variables pueden ser declaradas usando diversos tipos de datos; la plataforma Arduino incluye los siguientes:



byte es un tipo de dato que almacena un valor entero positivo de 8 bits (sin puntos decimales). Tienen un rango de 0 a 255 y también corresponde al tipo de dato unsigned char. Como ejemplo, considere:

byte algunaVariable = 180; //tipo entero positivo de un byte.

char es un dato de tipo entero que ocupa un byte en memoria; el rango varía de -128 a 127. Los caracteres simples de una sola letra se asignan entre simples comillas: 'a', y para una cadena de caracteres (múltiples letras) se emplea string y quedan entre doble comillas "hola". Por ejemplo,

char letra_a='a' ;//la letra queda en medio de comillas simples. char ascii_a=65;// es el código ASCII para la letra a.

unsigned char es un tipo de dato char, pero en este caso corresponde a un entero positivo debido al modificador de tipo de dato unsigned. La longitud de bits para unsigned char es de un byte en memoria y con un rango de 0 a 255. También se puede utilizar el tipo de dato byte, ya que son equivalentes. Por ejemplo:

unsigned char midato=135;//es equivalente a **byte**. byte midato=135;//el tipo de dato **byte** es equivalente a **unsigned char**.

El manejo de cadenas de texto (**string**) se realiza mediante arreglos o apuntadores de la siguiente forma (ver sección 4.5 **Arreglos**, página 85): char cadena[12];//un arreglo de 12 elementos tipo char, sin inicializar.

4.3 Variables 65

Las siguientes declaraciones determinan formas distintas para definir un arreglo tipo char para la cadena "robot" . El elemento final de los arreglos cadenaA[5] y cadenaB[6], respectivamente contienen el elemento nulo (NULL) e indican el fin del arreglo. Observe el uso de llaves { }, comillas y comas para asignar y referenciar a los elementos individuales de la cadena. El primer elemento del arreglo inicia con el pivote 0, posteriormente 1, 2, así sucesivamente hasta llegar al último elemento del arreglo, el cual no lleva coma antes de cerrar la llave }.

 $char \; cadena A[5] = \{ \; 'r' \; , 'o' \; , 'b' \; , 'o' \; , 't' \; \}; // cadena \; de \; cinco \; elementos \; tipo \; char. \\ char \; cadena B[6] = \{ \; 'r' \; , 'o' \; , 'b' \; , 'o' \; , 't' \; , \; ' \setminus 0' \; \}; // elemento \; nulo \; (NULL=' \setminus 0' \;).$

Las siguientes sentencias son declaraciones válidas para cadenas de texto:

char cadenaC[]={ "robot" };//se ajusta automáticamente al número de elementos.

char cadenaD[6]={ "robot" };//las comillas "" también ocupan espacio.

char cadenaE[9]={ "robot" };//se ocupan 5 elementos y 4 están disponibles.

El número de elementos del arreglo debe ser mayor o igual al número de caracteres (letras) de la cadena.

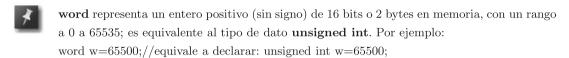
- short es un tipo de dato para números enteros de 16 bits o 2 bytes en memoria, con un rango de -2¹⁵ a 2¹⁵-1, es decir: -32,768 a 32,767. Ejemplo, short puerto1=127;//entero positivo. short codigoA=-23;//entero negativo.
- unsigned short corresponde a enteros positivos short sin signo, con 2 bytes en memoria y un rango de 0 a 65,535 (2¹⁶ 1). Por ejemplo: unsigned short codigoA=23;
- La palabra reservada **int** se emplea para declarar números enteros (sin puntos decimales) de longitud de 16 bits (2 bytes) y son datos primarios de la mayoría de microcontroladores. El rango de los números enteros es de -2¹⁵ a 2¹⁵-1, es decir: -32,768 a 32,767, por ejemplo: int i = 1500; //declara la variable i como tipo int. int j=-32004;//los enteros negativos son almacenados en complemento dos.

En algunas tarjetas Arduino como el modelo Due los números enteros ocupan 32 bits (4 bytes) en memoria, tendiendo un rango de -2,147,483,648 (-2^{31}) a 2,147,483,647 (2^{31}).

unsigned int corresponde al tipo de dato entero int sin signo, debido al modificador unsigned, con el siguiente rango de 0 a 65,535 (2¹⁶ - 1). Este tipo de dato es equivalente a word. Por ejemplo:

unsigned int k=63009;//equivale a declarar: word k=63009;

El modelo Arduino Due utiliza 32 bits (4 bytes) para representar un dato entero **unsigned** int, teniendo un rango de 0 a 4,294,967,295 (2^{32} - 1).



long corresponde al tipo de datos de tamaño extendido para enteros largos, sin puntos decimales; la longitud es de 32 bits y el rango abarca de -2,146,483,648 a 2,147,483,647. Ejemplo,

long j = 90000; //j es una variable tipo long de 16 bits o 2 bytes en memoria.

- unsigned long es un tipo de dato entero positivo largo (tipo long) sin signo, con 32 bits o 2 bytes en memoria, con un rango de 0 a 4,294,967,295 (2³²- 1). Ejemplo: unsigned long j1=3000000;//entero positivo largo de 32 bits sin signo.
- Los números reales IR se programan con la palabra clave **float**, representa un identificador para variables en punto flotante, cuya longitud corresponde a 32 bits para su representación dentro de un rango de -3.4028235E+38 a 3.4028235E+38. Un ejemplo de variable tipo flotante es el siguiente:

float x= 3.14, y=-0.0234; //declara x,y con valor inicial del tipo float.

double es un tipo extendido de dato float con doble precisión (utiliza 12 dígitos de precisión), y puede ocupar hasta 8 bytes o 64 bits en memoria. Sin embargo, para la mayoría de tarjetas Arduino ocupan 4 bytes o 32 bits. Ejemplo,

double x1=323456.67789;//variable flotante con doble precisión.

La tarjeta Arduino modelo Due maneja números flotantes con doble precisión de 64 bits o 8 bytes.

Arrays significan arreglos o registros de una colección de valores o datos que son accedidos con un pivote o índice numérico tipo entero positivo, siendo su primer elemento el índice cero, posteriormente 1, etc. Cualquier valor en el array debe llamarse escribiendo el nombre de la variable que corresponde a algún tipo de dato definido como boolean, char, byte, short, int, long, float, double y con el modificador unsigned para los enteros char, short, int y long.

El siguiente ejemplo muestra la declaración de una variable arreglo tipo entero y la asignación de valores a sus elementos:

int $u[] = \{0, -1, 4, 8\}; //se$ emplean $\{0, -1, 4, 8\}$ seguidos de una coma.

Otra forma para declarar un arreglo es definiendo el tamaño o número de elementos del arreglo:

int w[5];

Mayor información sobre arreglos, consulte la sección 4.5 Arreglos, página 85.

4.3 Variables 67

La tabla 4.2 contiene el resumen de tipos de datos que utiliza la plataforma Arduino y los rangos que cubren.

Tabla 4.2 Tipos de datos estándar para la plataforma Arduino.

| Tipo de datos | Longitud en memoria | Rango de valores |
|---|--|--|
| boolean bit1=true; boolean bit2=false; | 1 byte | true o false |
| <pre>char c='o' ; unsigned char d=123; unsigned char u[6]={"robot" }; char v[5]={'r' ,'o' ,'b' ,'o' ,'t' };</pre> | 1 byte 1 byte char u[6] 1 byte \times 6 char v[5] 1 byte \times 5 | -128 a 127 0 a 255 cada elemento: 0 a 255 cada elemento: -128 a 127 |
| byte byte1=123; | 1 byte = 8 bits sin signo | 0 a 255 |
| int i=234; | 2 bytes | -32768 a 32767 Algunos modelos Arduino como el Due los números enteros son de 32 bits o 4 bytes: $-2,147,483,648$ (-2^{31}) a $2,147,483,647$ (2^{31}). |
| unsigned int j=15345; | 2 bytes=16 bits sin signo | 0 a 65535 |
| word w=1890; | 2 bytes = 16 bits sin signo | 0 a 65535 |
| short s=156; | 2 bytes | -32768 a 32767 |
| long lg=-34567; | 4 bytes = 32 bits | -2147483648 a 2147483647 |
| unsigned long lgu=34567; | 4 bytes = 32 bits | 0 a 4294967295 |
| float x=123.458; | 4 bytes = 32 bits | -3.4028235×10 ³⁸ a 3.4028235×10 ³⁸ |
| double y=3.1416; | 4 bytes = 32 bits | -3.4028235×10^{38} a 3.4028235×10^{38} Números en punto flotante, doble precisión en el modelo Arduino Due ocupan 8 bytes (64 bits). |
| | 1 | Los arreglos pueden ser: boolean, byte, char, word, short, int, long, double, float y para los enteros char, int, short y long con el modificador unsigned. |



4.3.2 Modificadores de tipos de datos

Un modificador se utiliza para definir o declarar un tipo de dato, esto representa un fortaleza del lenguaje C ya que permite flexibilidad y ajustar el tipo de dato a determinadas necesidades de la aplicación.

Los modificadores que se emplean con los tipos de datos enteros son los siguientes:

signed.

unsigned.

long.

short.

const (también se aplica a float y double).

Los modificadores pueden combinarse con otros tipos de dato, para dar acceso a una variedad más amplia de variables. Por ejemplo, el modificador **long** se aplica con los números reales del tipo flotante con double precisión **double**, para obtener: **long double**.

Los modificadores **signed** y **unisgned** se pueden combinar con **signed int**, **signed short** int, **unsigned short** int, unsigned int, así como long int, unsigned long int, signed long y signed long int.

Es importante subrayar que los anteriores modificadores **signed**, **unsigned**, **short** y **long** no aplican con los tipos de datos **boolean**, **byte** y **word**; ya que son definiciones propias de Arduino y no pertenecen al lenguaje C.

4.3 Variables 69



Modificador de acceso const

El modificador de tipo **const** permite deshabilitar cualquier tipo de acceso a las variables indicadas.

Cuando se utiliza el modificador **const** hay que especificar el tipo de dato de las variables e inicializar en la parte donde se declara, el efecto que produce es que no permite acceder a la memoria de esa variable durante la ejecución del sketch, pero sí puede ser usada por cualquier tipo de expresión, sentencia y en operaciones aritméticas; por ejemplo, ${\bf const}$ float ${\bf g}{=}9.81;//{\rm aceleración}$.

Es de particular relevancia, la propiedad del modificador **const** en aplicaciones de robótica y mecatrónica, debido que estos sistemas contienen parámetros que no pueden ser modificados por ningún elemento de programación, es decir, el efecto real es proteger a los parámetros contra posibles errores de logística de programación.

Si en la fase de edición o escritura del sketch se inserta un código para cambiar el valor de la constante, por ejemplo g=0, entonces la fase de compilación marcará un error de sintaxis, advirtiendo al usuario que no lo puede realizar.

El modificador de acceso **const** protege contra escritura el área de memoria del dato declarado, es decir no permite cambiar el valor, permanece como una constante durante toda la ejecución del sketch.

El modificador **const** aplica también a los tipos de datos del sistema de programación Arduino **word, byte** y **boolean**.

Ejemplos de programación del modificador **const** son los siguientes:

```
const char f='0'; const boolean e1=false, e2=true; const byte b0=0, b1=1; const int paso=9; const signed int rp=-345; const unsigned int paso1=10; const word bandera=1; const short nab=234; const long lp=23456; const float pi=3.1416; const double ek=2.71;
```



Las anteriores sentencias indican el uso del modificador de acceso **const**; observe que todas las constantes declaradas se inicializan al momento de definirlas, de lo contrario el compilador indicará error de sintaxis.

La tabla 4.3 presenta los modificadores estándar del lenguaje C y sus características de longitud en número de bits o bytes y el rango específico que abarcan.

Tabla 4.3 Modificadores estándar para la plataforma Arduino.

| Modificador de tipo de datos | Longitud en memoria | Rango de valores |
|---|--|--|
| char unsigned char signed char | 1 byte | -128 a 127 0 a 255 -128 a 127 |
| int unsigned int signed int short int unsigned short int signed short int | 2 bytes | -32768 a 32767 0 a 65,535 -32768 a 32767 -32768 a 32767 0 a 65,535 -32768 a 32767 |
| short unsigned short signed short | 2 bytes | -32768 a 32767 0 a 65,535 -32768 a 32767 |
| long long int unsigned long unsigned long int signed long signed long int | 4 bytes | -2,146,483,648 a 2,147,483,647 -2,146,483,648 a 2,147,483,647 0 a 4,294,967,295 0 a 4,294,967,295 -2,146,483,648 a 2,147,483,647 |
| float | 4 bytes | -3.4028235E+38 a 3.4028235E+38 |
| double long double | 8 bytes 16 bytes | Longitud de 64 bits, aproximadamente 12 dígitos de precisión. Longitud de 128 bits, aproximadamen- te 24 dígitos de precisión. |
| const | Modificador de acceso, los bytes usados en me- moria dependen del ti- po de dato a modificar. | const es un modificador de acceso que permite mantener constante un tipo de dato. Se emplea con unsigned, signed, short para enteros (de 2 y 4 bytes) y long para flotante del lenguaje C, también se aplican a: boolean, byte y word. |

Todas las variables en el lenguaje C deben ser declaradas antes de ser usadas; se debe indicar el tipo de dato y modificadores (ver tablas 4.2 y 4.3, respectivamente).

Variables 71

& & Ejemplo 4.2

Desarrollar un sketch que defina o declare diversas constantes del tipo de datos: boolean, char unsigned char, byte, word, int, unsigned int, short, unsigned short, long, unsigned long, float y double. Desplegar sus respectivos valores numéricos en el monitor serial del ambiente de programación Arduino.

Solución

El skecth cap4_const tiene la finalidad de mostrar la forma de declarar y usar constantes de diferentes tipos de datos. El cuadro de código Arduino 4.2 contiene la descripción del sketch con la programación requerida en lenguaje C para visualizar su valor en el monitor serial del ambiente de programación Arduino. Para compilar, descargar el código de máquina del sketch y su ejecución en las tarjetas Arduino, consulte la página para ejecutar este sketch en las tarjetas Arduino consulte la página 61.

€ Código Arduino 4.2: sketch cap4_const

Arduino. Aplicaciones en Robótica y Mecatrónica.

Disponible en Web



Capítulo 4 Lenguaje C.

Fernando Reyes Cortés y Jaime Cid Monjaraz.

Alfaomega Grupo Editor: "Te acerca al conocimiento".

Sketch cap4_const.ino

- 1 const boolean bit0=false, bit1=true;//constante booleana de un byte (positivo).
- 2 const byte byte0=0, byte1=1;//constante entera de un byte (positivo).
- 3 const char f='0';//char de un byte
- 4 const unsigned char c1=123;//constante entera de un byte sin signo.
- 5 const signed char c2=-15;//constante entera de un byte
- 6 const word bandera=1000;//números enteros de 2 bytes
- 7 const short nab=234;//constante entera de 2 bytes.
- 8 const unsigned short s1=139;//constante entera sin signo de 2 bytes.
- 9 const signed short s2=-139;//constante entera de 2 bytes.
- 10 const unsigned short int s3=9050;//constante entera de 2 bytes.
- 11 const signed int i1=-9;//constante entera de 2 bytes.
- 12 const int i=0;//constante entera de 2 bytes.
- 13 const unsigned int i2=9;//constante entera sin signo de 2 bytes.
- 14 const short int i3=23000;//constante entera de 2 bytes.
- 15 const signed short int i4=-23000;//constante entera de 2 bytes.
- 16 const unsigned short int i5=9009;//constante entera sin signo de 2 bytes.

Continúa código Arduino 4.2a: sketch cap4_const Arduino. Aplicaciones en Robótica y Mecatrónica. Disponible en Web Capítulo 4 Lenguaje C. Fernando Reyes Cortés y Jaime Cid Monjaraz. Alfaomega Grupo Editor: "Te acerca al conocimiento". Continuación del sketch cap4_const.ino 17 const long j1=0x123abcd;//constante hexadecimal, enteros largos de 4 bytes. 18 const long int j2=-32000;//constante entera de 4 bytes. 19 const unsigned long j3=1234567;//constante entera sin signo de 4 bytes. 20 const unsigned long int j4=12345678;//constante entera de 4 bytes. 21 const signed long j5=-1234567;//constante entera de 4 bytes. 22 const signed long int j6=-134567;//constante entera de 4 bytes. 23 const float pi=3.141592;//número pi, punto flotante. 24 const double g=9.80665;//aceleración de la gravedad, constante flotante doble. 25 void setup(){ //función de configuración. 26 Serial.begin(9600);//inicializa comunicación serial 9600 Baudios. 27 } 28 void loop() { //lazo principal del sketch. Serial.print("Valor de la constante de un byte "); **2**9 Serial.print(byte1); //envía valor numérico al monitor serial. 30 Serial.print("\n");//nueva línea (retorno de carro). 31 Serial.print("Valor de la constante entera de 2 bytes"); **32** Serial.print(i4); //envía valor numérico al monitor serial. 33 Serial.print("\n");//nueva línea (retorno de carro). 34 Serial.print("Valor de la constante entera de 4 bytes"); 35 Serial.print(j4); //envía valor numérico al monitor serial. 36 Serial.print("\n");//nueva línea (retorno de carro). 37 38 Serial.print("Valor de la constante en punto flotante con 6 fracciones"); 39 Serial.print(pi,6); //envía valor numérico al monitor serial. Serial.print("\n");//nueva línea (retorno de carro). 40 Serial.println("Se repite el programa...."); 41 delay(5000); //Retardo de 5 segundos. 42 /para ejecutar este sketch en las tarjetas Arduino consulte la página 61.

4.3 Variables 73



4.3.3 Constantes para cadenas y de la plataforma Arduino

Las constantes en el lenguaje C se refieren a que necesariamente se tienen que inicializar en su declaración o predefinidos y posteriormente se manejan como valores fijos que no pueden ser alterados o cambiados por ninguna parte del programa.

Además de los tipos de datos enteros y flotante, hay otro tipo de constantes que se manejan para enteros del tipo **char** y cadenas (arreglo de caracteres). Las constantes tipo **char** se utilizan entre comillas simples, como por ejemplo: 'a' y las constantes para cadena van entre doble comillas ("Cuidado: robot activo"). La tabla 4.4 muestra algunos ejemplos de constantes para cadenas.

Tabla 4.4 Constantes para cadenas y formatos de números.

| Constantes | Descripción |
|------------|---|
| \0 | Nulo o fin de un apuntador (ver sección capítulo 5) |
| \t | Inserta tabulación horizontal |
| \\ | Barra invertida |
| \r | Salto de carro |
| \n | Inserta nueva línea |
| \b | Espacio atrás |
| \v | Inserta tabulación vertical |
| \f | Salto de página |
| \0 | Formato octal (base 8) |
| \x | Despliega números en formato hexadecimal (base 16) |
| Binario | Formato binario (base 2) |
| Decimal | Formato decimal (base 10) |

Las constantes definidas en la tabla 4.4 se utilizan cuando se despliega información en el monitor serial del ambiente de programación Arduino, así como en interfaz de la consola de comandos de MATLAB.

En la línea 17 del sketch cap4_const (ver cuadro de código Arduino 4.2) se utiliza la constante \0x

pegado al valor de la constante const
 long j1=0x123abcd para indicar que es una constante entera en formato hexadecimal. En la línea 31 se emplea un salto de línea n.

Tabla 4.5 Constantes de la plataforma Arduino.

| Constantes | Descripción | |
|--------------|--|--|
| true/false | false=0 y true=1 | |
| HIGH/LOW | Representan niveles lógicos: HIGH=1 (alto o 5 V) y LOW=0 (bajo o 0 V) | |
| INPUT/OUTPUT | Definen el modo de operación de los puertos digitales: INPUT entrada y OUTPUT salida. | |
| INPUT_PULLUP | Las tarjetas Arduino tienen de manera interna resistencias que están conectadas a la fuente interna (pull-up resistors, las cuales se pueden usar indicando en la función pinMO-DE(INPUT-PULLUP)). | |

Otra forma de definir constantes es por medio de la directiva #define de la siguiente forma:



#define estado 1//No se emplea punto y coma.



#define pi $3.1416//\text{constante }\pi$.

Note que cuando se utiliza #define no se emplea el operador punto y coma ; al final de la línea.

4.3.4 Ámbito de las variables

Las variables pueden ser declaradas al comienzo del programa antes de la función **setup()**, localmente dentro de funciones, y algunas veces en un bloque de declaración, por ejemplo en el interior de las instrucción **for**. Dependiendo del lugar donde la variable sea declarada determina el ámbito de la variable o la facultad de ciertas partes del programa para hacer uso de la variable.

Una variable global es aquella que puede ser vista y usada por cualquier función y declaración del programa. Las variables globales se declaran necesariamente al comienzo del programa, antes de la función **setup()**.

Una variable local es aquella que se define dentro de una función o como parte de una instrucción de lazo o control de programa como la instrucción for. El ámbito de la variable

4.3 Variables 75

local está restringido en la función o instrucción de lazo donde fue declarada, es decir sólo es visible y empleada dentro de ese ámbito.



Debe tenerse cuidado y no confundir como normalmente sucede con el ámbito de las variables globales y locales, es decir, puede haber dos o más variables del mismo nombre en diferentes sitios del programa que contienen diferentes valores. Si una variable local tiene el mismo nombre de una variable global, la variable local tiene prioridad en la función donde fue declarada, fuera de este ámbito la variable global es visible en todos los lugares del programa y la variable local no puede ser utilizada.

Ejemplos de variables globales y locales se ilustran en el código ejemplo 4.2: observe que en la primera línea del sketch se declaran variables enteras de tipo global $(\mathbf{i},\mathbf{j},\mathbf{k})$, las cuales son visibles en todo el programa. Sin embargo, en la rutina de configuración setup() existen variables enteras locales con el mismo nombre (i, j, k), bajo este escenario las variables locales tienen mayor prioridad sobre las variables globales, sólo cuando coinciden en tipo de dato y nombre del identificador, esto representa la única excepción de jerarquía que tienen las variables locales con respecto a las variable globales; por lo que, las variables globales no podrían ser manipuladas, bajo estas circunstancias. No es el caso para las variables x1 y x.

Código ejemplo 4.2

Variables globales y locales

```
int i,j,k; //variables globales, visibles por cualquier función y en cualquier parte del programa.
float x, y, z=2.345;
```

```
void setup(){
    int i,j,k; //cuando las variables locales coinciden en nombre y tipo de dato con las variables
    //globales, entonces las variables locales (i, j, k) tienen mayor jerarquía que sus homólogas
    //variables globales, y por lo tanto, no se podrían manipular en esta subrutina o función.
    i=0;
    j=9;
    k=i*j;
    x=33,4+z;
    y=x/j;
}
void loop(){
    float f; // la variable f sólo es visible dentro de loop().
    f = f + 10;
    x=f*10;
    y=f/100;
    z=f-100;
```



4.4 Operadores

E la lenguaje de programación C contiene un conjunto de operadores para manipular el valor de las variables.

Existen varios tipos de operadores como los que se describen a continuación:



Aritméticos (+, /, *, / y%)



Relacionales (>, <, ==, >=, <=)

Lógicos (&&, $||, \land \land$)

Manipulación de bits.

Hay otros operadores del lenguaje C como los siguientes:

Arreglos y apuntadores corchetes [], & y * (ver sección 4.5 y capítulo 5 Uniones, estructuras y apuntadores).

El operador ? sustituye a la instrucción condicional if()-else (ver sitio Web).

Operador coma, (ver página 59).

Punto y flecha . -> para uniones y estructuras (ver capítulo 5 Uniones, estructuras y apuntadores).

Los paréntesis () se usan en las funciones o subrutinas.

4.4.1 Operadores aritméticos

Los operadores aritméticos realizan operaciones con números enteros y en punto flotante; su forma de empleo, combinación y precedencia de uso es muy importante, ya que es clave en el valor numérico que resulta.

Los operadores aritméticos +, -, * y / se aplican a todos los tipos de datos definidos en lenguaje C; el operador % únicamente se aplica a tipos de datos enteros.

La tabla 4.6 presenta la descripción y notación de los operadores aritméticos.

| Operador aritmético | Notación | Ejemplo |
|---------------------|--|---------|
| Suma o adición | + | y=y+2; |
| Resta o sustracción | - | x=x-3; |
| Multiplicación | * | z=y*4; |
| División | / | w=w/5; |
| División en módulo | % sólo aplica a datos de tipo entero | y=15%4 |
| Incremento unitario | ++ incremen- ta en uno a su operando | x++; |
| Decremento unitario | decrementa en uno a su operando | x; |

Tabla 4.6 Operadores aritméticos.



Partes de la división

$$\frac{9}{2} = 4 \Leftrightarrow 2 \begin{vmatrix} \frac{4}{9} \\ 1 \end{vmatrix}$$

$$\text{divisor} \rightarrow 2 \mid \begin{array}{c} 4 \leftarrow \text{cociente} \\ 0 \leftarrow \text{dividendo} \end{array}$$

Dividendo es el número que se va a dividir, divisor es el número que divide, cociente es el resultado de la división, residuo o resto es lo que ha quedado del dividendo, que no se ha podido dividir por ser más pequeño que el divisor.

El operador división / se aplica a cualquier tipo de dato entero (byte, char, int, short, word y long, el residuo o resto de la división es truncado, es decir, no se asigna y por lo tanto no forma parte del resultado.

Por ejemplo, la división entera de los números 9 y 2: $\frac{9}{2} = 4$; el operador módulo % sólo se aplica a números enteros y obtiene el resto de la división entera, para este caso 9 %2=1 (no se puede aplicar a los números reales, por ejemplo a los tipos de datos **float** y **double**).

Incremento ++ y decremento --

Los operadores ++ y -- incrementa y decrementa en uno al operando, respectivamente. La sintaxis de estos operadores equivalen a los siguientes casos:



x++; //significa x=x+1;



x--; //significa x=x-1;

Los operadores ++ y -- no tienen una sintaxis única, pueden anteceder o estar posterior de su operando; los resultados son los mismos sobre el operando, pero cuando se utilizan en expresiones o sentencias se debe tener cuidado con la forma de utilizarlos, debido a que el resultado de asignación puede ser muy diferente.

La tabla 4.7 describe la sintaxis que pueden tener estos operadores:

Tabla 4.7 Operadores aritméticos ++ y --.

| Sentencia | Sintaxis equivalente | Sintaxis equivalente | |
|-----------|----------------------|----------------------|--|
| x=x-1; | x; | x; | |
| x=x+1; | x++; | ++x; | |

Importante: cuando los operadores ++ y -- se utilizan en expresiones existen diferencias notorias en la forma de trabajo y el valor numérico que proporcionan:

Cuando ++ y -- están antes o a la izquierda del operando, el lenguaje C lleva a cabo la operación aritmética de incremento o decremento, respectivamente, antes de usar el valor del operando. Por ejemplo, considere:

x=50;

y=++x;//primero se incrementa en uno a x y después se asigna el resultado a y.

z=100;

w=--z;//primero se decrementa en uno a z y después se asigna el resultado a w.

El resultado es: y=51; w=99; también x quedará con el valor de 51 y z con 99.



Cuando ++ y -- se encuentran posterior o después del operando, la operación aritmética respectivamente de incremento o decremento se realizará después de asignar el valor del operando. Por ejemplo, considere:

```
x=50; y=x++; //primero se asigna el valor 50 a \mathbf{y}, después se incrementa en uno a \mathbf{x}. z=100; w=z--;//primero se asigna el valor de 100 a \mathbf{w}, después se decrementa en uno a \mathbf{z}.
```

El resultado será de la siguiente forma:

```
y=50,
x tendrá el valor de 51,
w tendrá el valor de 100,
z quedará en 99.
```



La precedencia de los operadores ++ afecta el resultado cuando se emplean en expresiones; la jerarquía de los operadores es mayor cuando se encuentran a la izquierda de la variable y menor cuando están a su derecha.

Por ejemplo, $\mathbf{y}=++\mathbf{x}$; primero se incrementa en uno a \mathbf{x} y después se asigna el valor a \mathbf{y} ; para el caso $\mathbf{y}=\mathbf{x}++$; primero se asigna el valor de \mathbf{x} a la variable \mathbf{y} , después se incrementa en uno a \mathbf{x} .

Este proceso proporciona al usuario el control de programación, adaptando el código a múltiples aplicaciones.

Existe una forma de trabajar en lenguaje C con los operadores aritméticos (con excepción del operador %) para cualquier tipo de dato que facilitan la programación, la cual se denomina asignación compuesta y consiste en una contracción o forma resumida de sintaxis de los operadores aritméticos que se combinan entre ellos para realizar en forma más eficiente la asignación de resultados de una variable.

La tabla 4.8 muestra la forma de trabajo de operadores compuestos en su versión compacta, válidos para todo tipo de datos del lenguaje C.

Tabla 4.8 Combinación de operadores aritméticos.

| Sentencia | Sintaxis equivalente |
|-----------|----------------------|
| x++; | x=x+1; |
| ++x; | x=x+1; |
| x; | x=x-1; |
| x; | x=x-1; |
| y+=x; | y=y+x; |
| y-=x; | y=y-x; |
| y*=x; | y=y*x; |
| y/=x; | y=y/x; |

♣ ♣ Ejemplo 4.3

Desarrollar un sketch en lenguaje C donde se utilicen los operadores aritméticos para números enteros +, -, *, / y %, así como combinación entre ellos. Desplegar el resultado en el ambiente de programación Arduino.

Solución

El cuadro de código Arduino 4.3 contiene la programación en lenguaje C y descripción del sketch **cap4_operadores_arit**, ilustrando el empleo de los operadores aritméticos +, -, *, / y % en números enteros. La comunicación serial entre la computadora y la tarjeta Arduino queda establecida en 9600 Baudios de la rutina de configuración **setup()** (línea 3).

El lazo principal del sketch **loop()** inicia a partir de la línea 5. En las líneas 10 y 15 se programan los operadores aritméticos para incremento ++x y x++, respectivamente.

Los equivalentes operadores para sustracción (decremento) se encuentran respectivamente en las líneas 25 y 30. Los operadores combinados se ilustra su aplicación de las líneas 35 a la 48. La forma de empleo para el operador módulo % (entrega el residuo de la división entera) se presenta en la línea 52.

En la línea 54 se utiliza la función **delay**(10000) para generar intervalos de 10 segundos de pausa, tiempo que tarda cada ejecución del programa; los resultados son desplegados en el monitor serial del ambiente de programación Arduino.

```
€ Código Arduino 4.3: sketch cap4_operadores_arit
  Arduino. Aplicaciones en Robótica y Mecatrónica.
                                                       Disponible en Web
  Capítulo 4 Lenguaje C.
  Fernando Reyes Cortés y Jaime Cid Monjaraz.
  Alfaomega Grupo Editor: "Te acerca al conocimiento".
   Sketch cap4_operadores_arit.ino
 1 int i=16, j=3,y,x=50;//variables globales.
 2 void setup() { //rutina de configuración setup().
       Serial.begin(9600);//comunicación serie en 9600 Baudios.
 4 }
 5 void loop() { //rutina principal loop().
       Serial.println("Ejemplo de operadores x++, ++x"); //incremento x++, ++x
       Serial.print("Valor de x=");
       Serial.println(x);//enviar al monitor serial x=50.
 8
       Serial.print("Incremento y=++x==> valor de y=");
9
       y=++x;//primero incrementa y luego asigna: y=51, x=51, equivale a y=x+1;
10
11
       Serial.println(y);//enviar el valor de y=51.
       Serial.print("Nuevo valor de x=");
12
       Serial.println(x);//desplegar nuevo valor de x=51.
13
       Serial.print("Incremento y=x++==> nuevo valor de y=");
14
       y=x++; //primero asigna y=51, y después incrementa en uno x=52.
15
       Serial.println(y)//envía y=52;
16
       Serial.print("Valor actual de x=");
17
18
       Serial.println(x);//valor actual de x=52.
       Serial.println("\v");//inserta espacio tabular vertical.
19
       //Decremento x-- y --x
20
       Serial.println("Ejemplo de operadores x--, --x " );
21
       Serial.print("Valor de x= ");
22
       Serial.println(x);//despliega el valor actual de x=52.
23
       Serial.print("Decremento y=-x ==> valor de y= ");
\mathbf{24}
       y=-x;//primero decrementa, después asigna, equivale a y=x-1.
25
       Serial.println(y);//envía el valor de y=51.
26
       Serial.print("Nuevo valor de x= ");
27
       Serial.println(x);// valor de \mathbf{x}=51.
```

Continúa código Arduino 4.3a: sketch cap4_operadores_arit

Arduino. Aplicaciones en Robótica y Mecatrónica.



Capítulo 4 Lenguaje C.

Fernando Reyes Cortés y Jaime Cid Monjaraz.

Alfaomega Grupo Editor: "Te acerca al conocimiento".

Continuación del sketch cap4_operadores_arit.ino

```
Serial.print("Decremento y=x-==> valor de y=");
29
       y=x-;//primero asigna y=51, después decrementa x=51-1=50.
30
       Serial.println(y);//despliega el valor y=51.
31
32
       Serial.print("Valor actual de x=");
       Serial.println(x);//despliega el valor de \mathbf{x}=50.
33
       Serial.print("Combinación de operadores y += x; ==> y=y+x");
34
       y +=x;//equivale a y=y+x;
35
       Serial.println(y);//despliega el valor y=51+50=101.
36
       Serial.println("\v");//inserta espacio tabular.
37
38
       Serial.print("Combinación de operadores y -= x; ==> y=y-x");
39
       y = x; //equivale a y = y - x;
       Serial.println(y);//despliega y=101-50=51.
40
       Serial.print("Combinación de operadores y *= x; ==> y=y*x");
41
       y = x;//equivales a y=y*x;.
42
       Serial.println(y);//despliega el resultado y=51*50=2550.
43
       Serial.println("\v");//inserta espacio tabular vertical.
44
45
       Serial.print("Combinación de operadores y /= x; ==> y=y/x");
       y /=x;//equivale a y=y/x;.
46
47
       Serial.println(y);//despliega el resultado y=2550/50=51.
       Serial.println("\v");//inserta espacio vertical en blanco.
48
49
       Serial.print("División entera i/j= ");//División entera.
50
       Serial.println(i/j);//división entera: resultado 16/3=5.
       Serial.print("Residuo i %j= ");//resultado 1.
52
       Serial.println(i%j);//calcula el residuo de la división entera: 1.
53
       Serial.println("Se ejecuta una vez más el sketch en 10 segundos..." );
54
       delay(10000); //10 segundos.
55
       Serial.println("\v\v\v" );//inserta 4 espacios tabulares verticales en blanco.
56 }
57 //Para ejecutar este sketch en las tarjetas Arduino consulte la página 61.
```

Operadores de comparación

Los operadores de comparación o relacionales determinan un valor **verdadero** o **falso** dependiendo de la relación que exista entre dos tipos de datos (variables o constantes). La tabla 4.9 muestra la terminología y notación de los operadores de comparación.

Tabla 4.9 Operadores de comparación o relacionales.

| Notación | Descripción | |
|--|--------------------------------------|--|
| x==y | ${f x}$ es igual a ${f y}$ | |
| x!=y | ${f x}$ no es igual a ${f y}$ | |
| x <y< td=""><td>${f x}$ es menor que ${f y}$</td></y<> | ${f x}$ es menor que ${f y}$ | |
| x>y | ${f x}$ es mayor que ${f y}$ | |
| x >=y | ${f x}$ es mayor o igual que ${f y}$ | |
| x <=y | ${f x}$ es menor o igual que ${f y}$ | |



Resulta un error gramatical o error de sintaxis pretender usar lo siguiente: => y =<, ya que no existen como operadores en el lenguaje C.

Los operadores de comparación son ampliamente utilizados en las instrucciones de programación if() $\{...\}$, if() $\{...\}$ else $\{...\}$ for(;;) $\{...\}$, while() $\{...\}$ y do $\{...\}$ while();. Dependiendo si la condición es verdadera o falsa se realiza una determinada acción.

Operadores lógicos

Los operadores lógicos se emplean como una forma de comparar dos expresiones, devuelven un valor 1 o **true** y 0 o **false** dependiendo del tipo de operador lógico. La tabla 4.10 muestra los operadores lógicos básicos del lenguaje C y la tabla 4.11 muestra su tabla de verdad.

Estos operadores se usan a menudo en álgebra booleana y en declaraciones con las instrucciones condicionales: if() $\{...\}$, if() $\{...\}$ else $\{...\}$ y de lazo for(;;) $\{...\}$, while() $\{...\}$ y do $\{...\}$ while().

Tabla 4.10 Operadores lógicos básicos.

| Operador | Notación |
|----------|----------|
| and | && |
| or | |
| not | ! |

Tabla 4.11 Operadores lógicos y tabla de verdad.

| a | b | && | | !a |
|---|---|----|---|----|
| 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 |



Operadores a nivel de bits

En el sitio Web del libro, el lector puede consultar ejemplos adicionales en lenguaje C que incluyen manipulación con operadores lógicos a nivel de bits para los tipos de datos enteros (con sus variantes tales como: char, byte, word, int y long), así como operaciones para corrimiento a la izquierda y a la derecha.



Operadores lógicos y de comparación

Visite el sitio Web de esta obra para consultar ejemplos adicionales con operadores de comparación, así como operadores lógicos formando funciones booleanas (álgebra de Boole), principios propiedades y axiomas más importantes se implementan en lenguaje C y la ejecución del sketch en las tarjetas Arduino. Este material incluye ejemplos para manipular datos de tipo entero y a nivel de bits.

4.5 Arreglos 85

4.5 Arreglos



N arreglo o array es una colección de variables del mismo tipo que se declara, por ejemplo del tipo char, entero, flotante, etc.; a un elemento específico del arreglo se accede mediante un índice especificado por un número entero positivo.

La localidad de memoria asignada al arreglo consta de memoria continua, siendo la dirección de memoria más baja el primer elemento y la dirección más alta corresponde al último elemento; en el lenguaje C, los arreglos pueden ser de una o varias dimensiones.



4.5.1 Arreglos unidimensionales

La forma básica para declarar arreglos unidimensionales es especificando el tipo de dato, identificador o nombre del arreglo, el número de elementos encerrado por corchetes, operador coma, si se trata de una lista de arreglos y se finaliza con el operador punto y coma.

Por ejemplo: int band[3]; indica que es un arreglo de tres elementos de tipo entero. Otros ejemplos son:

```
int band[3]=\{0,-4,10\}, pin[5], pout[8];//arreglos de números enteros positivos. char comando[1]=\{'a'\}, led[11]=\{'RobotActivo''\};//arreglos de caracteres. float pos[100], vel[50], torque[40];//arreglos de números reales.
```

Igual como sucede en la declaración de las variables, los arreglos se pueden inicializar, en este caso abre una llave { seguido de la lista de elementos separados por comas, después del último elemento del arreglo no lleva coma, únicamente cierra lleva } y finalmente el operador punto y coma.

El tipo de datos válidos para los arreglos son los indicados por el lenguaje C más los definidos por la plataforma Arduinio **byte, boolean** y **word**. El resumen de los tipos de datos se presentan en la tabla 4.3, de esta forma el tamaño total de bytes que ocupa un arreglo depende del tipo de dato por la longitud (número de elementos del arreglo), es decir, considere el siguiente arreglo: float pos[100]; es un arreglo de 100 elementos de tipo flotante (4 bytes por elemento), desde pos[0] hasta pos[99], pos[100] no es precisamente un elemento del arreglo, más bien indica elemento nulo o fin del arreglo pos[100]=NULL. Para este caso el número total de bytes para el arreglo pos[0] es: 4 bytes pos[0] bytes de memoria.

Sea el siguiente arreglo de números reales: float vel[100];



El primer elemento de un arreglo se accede con el índice cero, vel[0];



El segundo elemento con el índice 1: vel[1]

Así sucesivamente, hasta el índice n-1, v[99], donde n=100 es el número total de elementos.

Para acceder a un arreglo no utilizar índices negativos v[-1], número reales o flotantes v[3.33], ya que representan errores de sintaxis.

Por ejemplo, considere una cadena de caracteres asignada al arreglo led, es decir: char led $[11]=\{$ 'RobotActivo" $\}$; el primer elemento es led[0]="R" , el segundo elemento led[1]="o" , el último elemento es led[10]="o" , puesto que led[11]=NULL; es decir, el elemento nulo indica que finaliza el arreglo led de números tipo char. Otra forma más simple de asignar la cadena de caracteres es sin llaves:

$$char led[11] = "RobotActivo";$$

4.5.2 Arreglos bidimensionales

Un arreglo bidimensional es la forma más simple de un arreglo multidimensional, el tipo de dato para esta clase de arreglos es cualquiera de los indicados en la tabla 4.3. La forma para declarar un arreglo es de la siguiente manera:

int mediciones[2][3];//arreglo bidimensional de 2 renglones por 3 columnas.

Observe que para especificar la dimensión se utilizan corchetes; un arreglo bidimensional obedece a una matriz de dimensión renglones \times columnas, el primer índice indica la dimensión de las filas o renglones, y el segundo indica el número de columnas. En este caso el arreglo mediciones tiene 2 renglones por 3 columnas.

Para inicializar un arreglo bidimensional se procede de la siguiente forma:

4.6 Funciones 87

Otra forma de realizar la declaración e inicialización es la siguiente:

int mediciones[2][3]= $\{0, -4, 6, //\text{rengl\'on} 0, \text{columnas } 0, 1 \text{ y } 2.$ 8, 6, -88, //rengl'on 1, columnas 0, 1 y 2. $0, 1, -3\}; //\text{rengl\'on} 2, \text{columnas } 0, 1 \text{ y } 2.$

Sea el siguiente arreglo de números reales: float vel[2][3];



El primer elemento de un arreglo bidimensional se accede con: vel[0][0];



El segundo elemento del arreglo bidimensional es vel[0][1], tercer elemento vel[0][2], así sucesivamente, hasta el último elemento vel[1][2].

Para acceder al arreglo bidimensional no usar índices negativos vel[-1][-3], número reales o flotantes vel[3.33][0.1], representan errores de sintaxis.

Arreglos multidimensionales

Los arreglos de más de dos dimensiones, se les denomina multidimensionales, por ejemplo, considere la siguiente declaración de un arreglo tridimensional: int activa_puertos[2][3][2]; es un arreglo de dos matrices de dimensión $\mathbb{R}^{3\times 2}$; se manejan en forma parecida a los arreglos bidimensionales. Véase el ejemplo 4.7, el cual contiene una aplicación de arreglos.

4.6 Funciones



As funciones también conocidas como **subrutinas**, subprogramas, rutinas o procedimientos son recursos importantes de programación del lenguaje C; contienen grupos de sentencias y declaraciones para formar bloques de código, ejecutan tareas específicas, facilitan el diseño, mantenimiento y depuración de código; dan claridad a la forma de programar y representan una técnica sencilla de automatizar aplicaciones en diversas áreas de la ingeniería. Los tipos de funciones básicos del lenguaje C se muestran en la tabla 4.12.

La sintaxis de una función consiste en declarar el tipo de resultado que devuelve la función, el cual corresponde a los tipos de datos convencionales del lenguaje C (ver tabla 4.12), continúa posteriormente al menos con un espacio en blanco y luego el identificador o nombre de la función, continúa entre paréntesis la lista de parámetros de entrada, separados por comas e indicando el tipo de datos (con excepción de **void**, el cual es una extensión del estándar de ANSI), prosigue con llave de apertura {. Como parte del código de la función se encuentra la declaración de variables locales,

constantes y conjunto de sentencias; cuando dicha función no es del tipo void, generalmente se emplea la palabra clave **return** para devolver el resultado y finaliza con cierre de llave }. El código ejemplo 4.3 muestra los elementos básicos de programación de una función.

Código ejemplo 4.3

Sintaxis de una función

```
tipo_de_dato
               nombre_de_la_función(lista de parámetros){ //se abre llave
    //La lista de parámetros de la función se encuentra delimitada por paréntesis.
    //Cada elemento de la lista de parámetros está separado por comas,
    // definiendo el tipo de dato (void no se aplica a la lista de parámetros).
    Declaración de variables locales de la función;//void no aplica a variables.
    Grupo de sentencias;
    //La palabra clave return se emplea para devolver el resultado de la función.
    return 1 //Cuando la función es del tipo void no se emplea return.
} //la función finaliza con cierre de llave.
```



Sintaxis de funciones en lenguaje C

Se han preparado ejemplos ilustrativos con las funciones Arduino para su mejor comprensión y aplicación. Estos ejemplos didácticos se encuentran disponibles en el sitio Web de este libro. Se invita al lector que los descargue, estudie y ejecute en los diversos modelos de tarjetas Arduino.

Ejemplos de funciones

El uso de rutinas de servicio de interrupciones, manejo de interrupciones externas e internas son temas complejos que merecen ser abordados en detalle. En el sitio Web de la presente obra dentro del capítulo 12 "Manejo de interrupciones" se han preparado ejemplos con interrupciones con aplicaciones en control automático y en tiempo real. Se invita al lector que descargue esta información para mejorar sus conocimientos de programación y control del sistema Arduino.

4.6 Funciones 89

Tabla 4.12 Tipos básicos de funciones.

| Tipo de datos | Descripción |
|----------------------|--|
| boolean | Booleano. |
| char | Char. |
| byte | Entero positivo de un byte. |
| int | Entero de 2 bytes. |
| unsigned int | Entero positivo o sin signo de 2 bytes. |
| word | Entero de 2 bytes. |
| long | Entero largo de 4 bytes. |
| unsigned long | Entero largo sin signo de 4 bytes. |
| float | Número real en punto flotante de 4 bytes. |
| double | Número real en punto flotante con doble precisión de 4 bytes. |
| void | Ningún tipo de dato retorna. |
| apuntador y arreglos | Tipos de datos válidos del lenguaje C definidos como arreglos y apun- tadores. |



Además de los tipos de datos convencionales del lenguaje C, una función también puede ser del tipo **void**, significa que no retorna algún tipo de dato, esta característica es específica de las funciones y no se aplica a variables ni a constantes. En el sistema Arduino las funciones **setup()** y **loop()** son ejemplos del tipo **void** y no llevan parámetros de entrada.



Una función puede ser definida sin parámetros de entrada, pero sí requiere el uso de parétesis () como: setup() y loop().

Las funciones también pueden ser del tipo apuntador y arreglos (mayores detalles ver el capítulo 5 Uniones, estructuras y apuntadores).

El código ejemplo 4.4 muestra la forma de programar la norma de un vector $x \in \mathbb{R}^3$, esta función es del tipo **float**, es decir devuelve la norma de un vector de dimensión 3 como un número real en punto flotante. Entre paréntesis se declaran los argumentos de la función, que en este caso corresponden a los componentes del vector $x = [x_1, x_2, x_3]^T$ como números flotantes y el identificador **norma_x** es declarado dentro de la función **norma_euclidiana** como variable local.

La función **norma_euclidiana** retorna el resultado a través de la sentencia **return**, esto es necesario debido a que técnicamente **norma_x** es una variable local; si esta variable se define como variable global, entonces es opcional emplear **return**.

float norma_euclidiana(float x1, float x2, float x3){....}

La sintaxis de una función tiene los siguientes componentes:



Tipo de dato que devuelve, para este ejemplo **float**, el identificador o nombre de la función, en este caso **norma_euclidiana**, entre paréntesis la lista de argumentos o parámetros de entrada de la función indicando el tipo de dato de ellos, cada argumento deberá estar separado por una coma: (float x1, float x2, float x3); además, se usan llaves {....} para delimitar el código correspondiente a la función.



Cuando el resultado que retorna una función se registra en alguna de sus variables locales, entonces hay que emplear la sentencia **return**, por ejemplo: return norma_x;, siendo norma_x una variable de ámbito local.

Código ejemplo 4.4

Norma de un vector $\boldsymbol{x} \in \mathbb{R}^3$

```
float norma_euclidiana(float x1, float x2, float x3){

float norma_x;//variable local definida como número real en punto flotante.

//Cálculo de la norma euclidiana: ||x|| = \sqrt{x_1^2 + x_2^2 + x_3^2}

norma_x=sqrt(x1*x1+x2*x2+x3*x3);

return norma_x;//retorna la norma euclidiana del vector x \in \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}.
```

Observe que se emplea la función raíz cuadrada $\mathbf{sqrt}(\mathbf{x1*x1+x2*x2+x3*x3})$ con parámetro de entrada $x_1^2 + x_2^2 + x_3^2$ que corresponde a la sumatoria de los componentes al cuadro del vector \boldsymbol{x} .

Una función se distingue por el uso de paréntesis (), esto es un rasgo distintivo que caracteriza a todas las funciones y no siempre llevan parámetros dentro de los paréntesis, en tal caso depende de la definición y diseño de la función.

4.6 Funciones 91

return

La sentencia **return** fuerza la terminación de la ejecución del código de la función y devuelve el resultado esperado. En el ejemplo 4.4 se requiere **return**, debido a que la variable **norma_x** es local a la función **norma_euclidiana** de otra no podría hacerlo, a menos que la variable **norma_x** sea global, entonces **return** resulta opcional.

♣ Ejemplo 4.4

Calcular la norma euclidiana del siguiente vector: $\mathbf{x} = \begin{bmatrix} 1.23 \\ 3.34565 \\ -45.678 \end{bmatrix}$, enviar el resultado al monitor serial del ambiente Arduino.

Solución

El sketch **cap4_normavector** tiene la programación en lenguaje C para calcular la norma euclidiana del vector $\boldsymbol{x} = \begin{bmatrix} 1.23 & 3.34565 & -45.678 \end{bmatrix}^T \in \mathbb{R}^3$ y desplegar el resultado en el monitor serial del ambiente de programación Arduino. La descripción y documentación de este sketch se encuentra en el cuadro de código Arduino 4.4.

En la línea 1 se encuentra la función o subrutina **norma_euclidiana**, la cual es declarada como una función de tipo **float** y tiene argumentos o parámetros de entrada los componentes del vector $\mathbf{x} = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix}^T$, esta función contiene a la variable **morma_x**, la cual es local a dicha función, el resultado se devuelve con la sentencia **return x**;. Observe que como en la línea 6 finaliza la función **norma_euclidiana** con cierre de llave }.



La sintaxis de la sentencia **return** puede aceptar paréntesis para delimitar a la variable que retorna el resultado, por ejemplo: en la línea 1 se utiliza **return norma_x**; pero también es válido **return(norma_x)**;

También puede llevar uno o más espacios en blanco entre la palabra clave **return** y el paréntesis que abre, por ejemplo: **return** (**norma_x**);

En la línea 8 se programa la comunicación serial USB en 9600 Baudios, dentro de la función tipo void setup(); en la línea 10 empieza la función o subrutina principal del sketch loop() declarada como tipo void. Otras funciones que se emplean son: Serial.print(...) acepta argumentos de entrada cadena de caracteres o datos flotantes con el número de fracciones a desplegar (ver líneas 14 y 15, respectivamente), la línea 23 contiene a la función delay(3000), la cual acepta un número entero positivo para generar un retardo o pausa de 3 segundos.

```
○ Código Arduino 4.4: sketch cap4_normavector
  Arduino. Aplicaciones en Robótica y Mecatrónica.
                                                       Disponible en Web
  Capítulo 4 Lenguaje C.
  Fernando Reyes Cortés y Jaime Cid Monjaraz.
  Alfaomega Grupo Editor: "Te acerca al conocimiento".
   Sketch cap4_normavector.ino
 1 float norma_euclidiana(float x1, float x2, float x3){
       float norma_x;//variable local definida como número real en punto flotante.
       //Cálculo de la norma euclidiana.
3
       norma_x = sqrt(x1*x1+x2*x2+x3*x3);
       // return(norma_x);//este código es válido, return acepta esta sintaxis.
   return norma_x;//retorna la norma euclidiana del vector x.
 6 }
7 void setup(){ //función de configuración.
       Serial.begin(9600);//velocidad de transmisión serial 9600 Baudios.
9 }
10 void loop() { //función o subrutina principal loop().
       float resultado; //variable local a loop()
11
       float x1=1.23, x2=3.34565, x3=-45.678;//componentes del vector x.
12
13
       Serial.println("Cálculo de la norma euclidiana de un vector x dimensión 3.");
       Serial.print("Componente x1="); //envía letrero
14
       Serial.println(x1,4); //envía valor numérico con 4 fracciones.
15
16
       Serial.print("Componente x2=");
17
       Serial.println(x2,4);
18
       Serial.print("Componente x3=");
       Serial.println(x3,4);
19
       resultado=norma_euclidiana(x1, x2, x3);
20
       Serial.print("Norma euclidiana = ");
       Serial.println(resultado,4);
22
23
       delay(3000); //genera pausa de 3 segundos.
       Serial.println("En 3 segundos inicia una vez más rutina principal loop(){...}.");
\mathbf{24}
25 }
   //Para ejecutar este programa consulte el procedimiento de la página 61.
```

La función **norma_euclidiana** (línea 1) puede estar colocada por arriba de la función **void setup()**,

4.6 Funciones 93

tal y como está programada en el cuadro de código Arduino 4.4; otra opción es estar después de la llave de cierre de la subrutina **void setup()** y antes de la función **void loop()** o también justo por debajo de la llave de cierre de la misma función **void loop()** (después de la línea 25). La compilación, descargar del código de máquina y la ejecución del sketch **cap4_normavector** se describe en el procedimiento sugerido en la página 61.



Las funciones o subrutinas no tienen un lugar específico dentro del programa, pueden ir antes o después de la función principal **void loop()** pero no dentro de esta función.



El lenguaje C no permite la creación de funciones dentro de funciones.

Tome siempre en cuenta que el uso excesivo de variables globales puede provocar errores en el programa, siempre será mejor diseñar funciones con sus variables locales necesarias de tal forma que el comportamiento de la función en la medida de lo posible no dependa de variables globales o identificadores externos.

Librerías

Las librerías son archivos de enlace dinámico (**dynamic-link library**) o bibliotecas con extensión **dll** que incluyen el código ya compilado de un conjunto de funciones que realizan una tarea específica, generalmente las librerías van acompañadas por un archivo tipo cabecera o header como parte del skecth que indican como escribir correctamente los identificadores o nombres de las funciones, como por ejemplo para las librerías estándar de puerto I/O se utiliza **stdio.h**, para funciones matemáticas **math.h**, para el manejo de servos **Servo.h**, etc. Cuando el usuario escribe código de una función, y éste se encuentra depurado, entonces se puede compilar por separado y enlazar con otro conjunto de funciones para utilizarse posteriormente sin tener que utilizar el código fuente. Mayor información sobre bibliotecas ver el capítulo 6 **Librerías y funciones Arduino**.

Técnicamente una biblioteca o librería es un conjunto de funciones compiladas por separado que se pueden enlazar con el programa del usuario.



4.7 Instrucciones de programación

E acuerdo con la sintaxis del lenguaje de programación C una sentencia de control de programa (instrucciones de programación) pueden ser del tipo condicional, control de bucle o lazos de programas.



Condicionales: **if**, **if-else** y **switch**.



Bucles o lazos de control: **for, while, do-while**. El lenguaje C tiene un conjunto de instrucciones que permiten que un bloque de sentencias se ejecute un número determinado de veces hasta alcanzar cierta condición.

break, continue y goto (esta última sentencia en sitio Web).

4.7.1 Instrucciones condicionales

El lenguaje de programación C contiene un conjunto de instrucciones del tipo condicional que toman un papel muy importante en aplicaciones de ingeniería robótica y mecatrónica.

A continuación se describe la sintaxis de las instrucciones condicionales:

```
 if (valor)\{...\} \\ if (valor)\{...\} else \{...\} \\ \\ if (valor1)\{...\} else if (valor2)\{...\}
```

Instrucción condicional if(valor) {...}

La instrucción condicional if tiene varias formas estructurales con sintaxis diferentes, dependiendo de la manera de emplearla. La estructura más simple corresponde a la forma: if(valor){...}, en este caso se evalúa la condición valor como verdadero o falso; si valor es verdadero, entonces se ejecuta el bloque de instrucciones delimitados por las llaves {...}; si valor es falso, continúa con las instrucciones que se encuentran después del delimitador de la llave de cierre }.

La figura 4.1 muestra el diagrama de flujo con estructura simple if(valor){...}, observe que

dependiendo del valor booleano que tenga **valor** se ejecuta el bloque de instrucciones (cuando **valor** es verdadero o entero positivo) o continúa con la secuencia del programa.

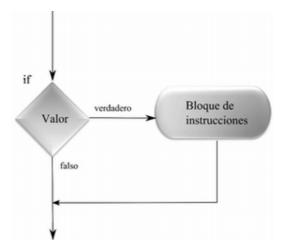


Figura 4.1 Estructura simple de la instrucción condicional if(valor){...}.

Para el caso donde la instrucción condicional **if(valor)** ejecute una sola instrucción es opcional usar llaves; generalmente, por estética en el estilo de programación del lenguaje C y sólo en este caso se pueden omitir, por ejemplo:



if (a==b) a=a+1; //con una sola instrucción no se requiere que se encuentre delimitada por llaves, es opcional.

Sin embargo, también es válido que el usuario delimite una sola instrucción con llaves:

if (a==b) {a=a+1;} //una sola instrucción puede estar delimitada por llaves, debido a que da certeza al programar, evitando errores de sintaxis.

Por otro lado, cuando la instrucción **if()** contenga un bloque con dos o más instrucciones, necesariamente deberán estar delimitadas por llaves {...}, por ejemplo:

if $(a==b){a=a+1; b=a*a;}$ //el bloque de instrucciones debe estar delimitado por llaves.

De no ser así, la instrucción **if** sólo ejecutará la primera sentencia a=a+1; y no realizará la operación **b=a*a**; ocasionando posibles problemas de logística para el resto del sketch.

Otros estilos válidos de programación del lenguaje C que se aplican a la instrucción **if**() $\{...\}$ son los siguientes:

Observe el uso de la sangría en cada renglón por debajo de la instrucción **if** que indica el bloque de instrucciones que pertenecen a dicha instrucción condicional.

Para el segundo y tercer caso de los ejemplos anteriores, la llave que cierra } debe estar alineada con la instrucción if. Estos son aspectos básicos de estilo y estética de programación del lenguaje C que ayudan a depurar la escritura del código y encontrar posibles errores de sintaxis.

♣ Ejemplo 4.5

Desarrollar un sketch en lenguaje C que permita detectar el número 12 del conjunto de valores que adquiere una variable que se incrementa unitariamente.

Solución

Como una aplicación sencilla de la instrucción condicional **if(valor)**{...} con estructura simple se encuentra el código 4.5 del sktech **cap4_if**. La variable tipo entero **i** se declara global y se inicializa en cero (ver línea 1).

La rutina **setup()** programa la comunicación USB con la computadora en 9600 Baudios como se indica en la línea 3. La rutina principal **loop()** inicia en la línea 5 con el código en lenguaje C de la aplicación requerida.

⊙ Código Arduino 4.5: sketch cap4_if Arduino. Aplicaciones en Robótica y Mecatrónica. Disponible en Web Capítulo 4 Lenguaje C. Fernando Reyes Cortés y Jaime Cid Monjaraz. Alfaomega Grupo Editor: "Te acerca al conocimiento". Sketch cap4_if.ino 1 int i=0;/*inicializa variable en cero.*/ 2 void setup(){ Serial.begin(9600); //inicializa comunicación serial 9600 Baudios. 4 } 5 void loop() { //rutina principal: loop(). i++; /*incrementa contador. */ /*estructura simple de la instrucción if(valor){...}*/ if(i==12) i=0; /*determina el valor lógico (falso o verdadero) de i==2. */ Serial.println(i); //envía valor numérico al monitor serial. 10 delay(100); //Retardo de cien milisegundos. 11 }



Instrucción if

La instrucción condicional if es de las más utilizadas en programación en lenguaje C, en el sitio Web de esta obra se han preparado un conjunto de programas o sketchs didácticos que tienen la finalidad de ilustrar su sintaxis, comprensión y aplicación. Se invita al lector que descargue dichos ejemplos, compile y ejecute en las tarjetas Arduino.

Ejemplos con if

En el sitio Web de este libro se encuentran disponibles varios ejemplos didácticos de la instrucción if, incluyen las diversas variantes de sintaxis que puede presentar esta instrucción condicional, combinando con else, y también con la versión anidada de instrucciones if. Se presentan aplicaciones que determinan el estado de interruptores electrónicos y mecánicos, sensores, niveles de voltaje, entre otros más.

Para ejecutar el sketch **cap4_if** en las tarjetas Arduino, realizar lo siguiente: dentro del ambiente de programación Arduino cargar el código 4.5 (disponible del sitio Web del libro web), seguir los pasos requeridos que se describen en la página 61.

Dentro de la declaración de la instrucción **if** en la línea 8 debe tener cuidado con el uso del operador == y no confundirlo con el signo de asignación =.

Instrucción condicional if(valor) $\{...\}$ else $\{...\}$

Dentro de las variantes de sintaxis que tiene la instrucción condicional **if(valor)** se encuentra la siguiente estructura gramatical: **if(valor)** {bloque 1} else{bloque 2}. Si valor adquiere una condición verdadera (**true**, cualquier valor diferente a cero) se ejecuta el bloque 1 de instrucciones que se encuentra delimitado por las llaves de la instrucción **if**; en el caso que valor sea una condición falsa (**false** o cero), entonces se ejecuta el bloque 2 delimitado por las llaves correspondientes a la palabra clave **else**.

El funcionamiento cualitativo de la instrucción **if(valor)** {bloque1;} else{bloque2;} es que se ejecuta uno de los dos bloques (no ambos). En la figura 4.2 se describe el diagrama de flujo de la instrucción **if(valor)** en con estructura gramatical: **if(valor)** {...} else{...}.

♣ Ejemplo 4.6

Desarrollar un sketch que permita clasificar números enteros pares e impares de una lista de valores comprendidos en forma consecutiva entre 0 y 100; desplegar en el monitor el listado de valores e indicar la cantidad de números pares e impares.

Solución

Considere el cuadro 4.6 que ejecuta el código en lenguaje C del sketch $\mathbf{cap4_ifelse}$. Para analizar si un número es par o impar se utiliza el siguiente razonamiento: debido a que los números a procesar son del tipo entero comprendidos entre 0 y 100, al i-ésimo elemento del listado de valores se divide entre 2; si el residuo de la división es cero significa que es par, en otro caso, es impar $\mathbf{else\{...\}}$. Para realizar este procedimiento, el sketch $\mathbf{cap4_ifelse}$ utiliza la función $\mathbf{fmod(i,2)}$ que devuelve el residuo de la división entera $\mathbf{i/2}$ (línea 9); el uso de esta función requiere necesariamente insertar en la cabecera del sketch (header) la correspondiente librería de matemáticas donde se encuentra definido el protocolo de sintaxis de dicha función: #include < math.h>, ver la línea 2 (para mayores detalles sobre el manejo y uso de librerías ver capítulo 6 $\mathbf{Librerías}$).

El inicio del sketch **cap4_ifelse** consiste en declarar las variables de tipo entero que se utilizarán en el algoritmo a desarrollar. En la línea 3 se declaran las variables globales i, j, k; la variable i se

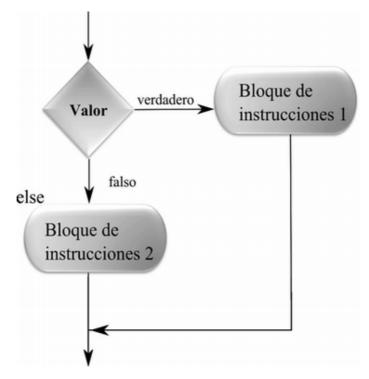


Figura 4.2 Instrucción condicional if(valor) {...} else{...}.

utiliza para incrementar de uno en uno los valores comprendidos entre 0 y 100 (en la línea 26 se envía la lista de valores de esta variable en forma permanente al monitor serial).

Note que en la línea 27 se emplean retardos de una décima de segundo entre datos para propósitos de estabilizar la comunicación USB, mientras que, los pivotes **j**, **k** se utilizan para contabilizar la cantidad de números pares e impares, respectivamente, así como registrar y direccionar los elementos de los arreglos correspondientes **par**[99] e **impar**[99] (definidos en la línea 4).

La rutina **setup()** programa la comunicación USB estableciendo la velocidad de 9600 Baudios (línea 6). La rutina principal **loop()** se ubica en la línea 8 con el código que determina el algoritmo para seleccionar números pares e impares, y envío de datos a la computadora donde radica el monitor serial del ambiente de programación Arduino.

En la línea 9 se encuentra la instrucción condicional **if**(**fmod**(**i**,**2**)==**0**), si es verdadera se ejecuta el bloque 1, compuesto por el registro de los números pares, así como contabilizar dichos números. Si la condición de la instrucción **if**(**fmod**(**i**,**2**)==**0**) toma un valor falso, se ejecuta el bloque 2 asociado a **else**{...}, iniciando el proceso de los números impares en la línea 13.

Cuando la variable i toma el valor máximo de 100, entonces en la línea 18 la condición if(i==100) es verdadera, se envían al monitor serial la cantidad de números pares e impares, inicializando a cero variable contador y pivotes para repetir una vez más el proceso de manera indefinida por la función loop(); lo anterior se le indica al usuario por medio de letreros con pausa de 5 segundos por medio de la función delay(5000), como se indica en la línea 24.

Dentro de la declaración de la instrucción **if**(**fmod**(**i**,**2**)==**0**) en la línea 9 también es posible programar al operador igualdad == como: **if**(**0**==**if**(**fmod**(**i**,**2**)), lo mismo sucede en la línea 18 **if**(**i**==**10**) puede ser programado como: **if**(**100**==**i**). Es decir, este tipo de operador es conmutativo y por lo tanto, no depende del orden de los operandos.

Para compilar y descargar el código de máquina desde el ambiente de programación Arduino y la ejecución del sketch **cap4_ifelse** en cualquier tarjeta de los modelos Arduino:



Consulte el procedimiento indicado en la sección 2.4 del capítulo 2 **Instalación y puesta** a punto del sistema Arduino.



También puede revisar la versión compacta de dicho procedimiento en la página 61.

Instrucción if() $\{...\}$ else $\{...\}$

La instrucción condicional if es de las más utilizadas en programación en lenguaje C, en el sitio Web de esta obra se han preparado un conjunto de programas o sketchs didácticos que tienen la finalidad de ilustrar su sintaxis, comprensión y aplicación. Se invita al lector que descargue dichos ejemplos, compile y ejecute en las tarjetas Arduino.

Instrucción if anidada

La instrucción if puede determinar el valor verdadero o falso que retorna una función específica, el análisis y discusión de este tipo de sintaxis se presenta a través de varios ejemplos didácticos contenidos en el sitio Web de esta obra, donde se han preparado código fuente que involucra la incorporación de la instrucción condicional if en forma anidada; se invita al lector a consultar dicho material adicional.

```
€ Código Arduino 4.6: sketch cap4_ifelse
  Arduino. Aplicaciones en Robótica y Mecatrónica.
                                                       Disponible en Web
  Capítulo 4 Lenguaje C.
  Fernando Reyes Cortés y Jaime Cid Monjaraz.
  Alfaomega Grupo Editor: "Te acerca al conocimiento".
   Sketch cap4_ifelse.ino
 1 /* Programa ejemplo de la instrucción if(valor){bloque 1} else{bloque 2} */
 2 #include <math.h> /*cabecera del programa: librería de funciones matemáticas. */
 3 int i=0, j=0, k=0; /*inicializa variables en cero. */
 4 int par[99], impar[99];/*apuntadores para registrar números pares e impares.*/
 5 void setup(){//función de configuración setup().
       Serial.begin(9600); //Inicializa comunicación serial 9600 Baudios.
 7 }
 8 void loop(){ //rutina principal: loop().
       if(fmod(i,2)==0){/*bloque 1: número par.*/
9
         par[j]=i;/*registra número par.*/
10
         j=j+1;/*incrementa pivote de números pares.*/
11
         \frac{1}{f(mod(i,2))} = 0
12
13
       else {/*bloque 2: número impar.*/
         impar[k]=i;/*registra número impar.*/
14
         k=k+1;/*incrementa pivote de números impares.*/
15
       }//finaliza else{..}.
16
       i++; /*incrementa valor de la variable i. */
17
       if(i==100) {
18
19
         Serial.println("Cantidad de números pares");
20
         Serial.println(j);
         Serial.println("Cantidad de números impares"); Serial.println(k);
21
22
         i=0; j=0; k=0;//reset para todos los pivotes.
23
         Serial.println("Inicia nuevamente el sketch" );
24
         delay(5000);//retardo por 5 segundos.
25
       26
       Serial.println(i);//envía al monitor serial el valor de i.
27
       delay(100); //retardo de cien milisegundos.
28 }//para ejecutar este sketch en las tarjetas Arduino, consulte la página 61.
```



4.7.2 Instrucción switch(valor) { case: ... break; default: ...}

La sentencia $\mathbf{switch(valor)}$ es una instrucción de decisiones múltiples, donde \mathbf{valor} es sucesivamente comparado con una lista de enteros constantes o caracteres ($\mathbf{case}\ \mathbf{1},\ \mathbf{case}\ \mathbf{2},\ \cdots$, $\mathbf{case}\ n$); cuando encuentra la correspondencia, entonces se ejecuta un bloque de instrucciones. La forma general de \mathbf{switch} se muestra en el código Arduino 4.5:

Código ejemplo 4.5

```
switch(valor) { case: ... break; default:...}
switch(valor){
    case 1:
       bloque de instrucciones 1;
       break;
    case 2:
       bloque de instrucciones 2;
       break;
    case n:
       bloque de instrucciones n;
       break;
    case n+1:
           /*bloque de instrucciones de case n + 1.*/
       break;
       default:
           /*generalmente la sentencia default (es opcional y no lleva la sentencia break),*/
           /*bloque de instrucciones de la sentencia default: */
              además, default precede a la llave de cierre } de la instrucción switch.*/
//Continúa el resto del programa.
```

Cuando se encuentra una correspondencia, el bloque de instrucciones asociado con el respectivo **case** se ejecuta hasta alcanzar la sentencia **break**; para terminar el flujo del programa de la constante **case** respectiva, saliendo de la instrucción condicional **switch** para continuar con las siguientes instrucciones del programa.

En el escenario de que no exista sentencia de salida **break**; (puesto que es opcional) el programa continúa con el siguiente **case**. Técnicamente **case** es una etiqueta que marca el inicio de la ejecución

Alfaomega

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

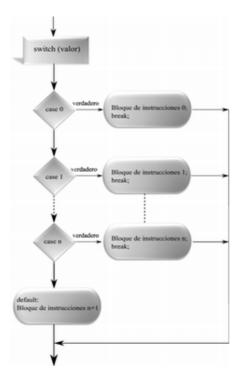
Fernando Reyes Cortés • Jaime Cid Monjaraz

de un bloque de instrucciones hasta encontrar una sentencia **break** o encuentre la llave de cierre } de la sentencia **switch**.

Cuando no se encuentre correspondencia con algún **case**, entonces se ejecuta el bloque de instrucciones asociado con **default**, siendo esta sentencia opcional; no es necesario usar a la sentencia **break** al final del bloque de instrucciones asociado con **default**, ya que no tendría sentido; **default** está destinado a considerar casos no previstos dentro de la logística de programación de la instrucción **swtich**().

Generalmente, **default** se coloca después del último **case**, y si no se encuentra presente ningún bloque de instrucciones a ejecutar, entonces termina la sentencia **switch**.

Para propósitos de ofrecer una explicación cualitativa sobre el funcionamiento y programación de la sentencia condicional switch(valor) { case: ... break; default: ... }, en la figura 4.3 se presenta el diagrama de flujo con la descripción completa de los diferentes escenarios que puede tener esta instrucción.





Ejemplos con el operador?

Una variante de sintaxis de la instrucción if(){} else{} es por medio del operador ?, el cual da mayor estética en el estilo de programación del lenguaje C. En el sitio Web de esta obra, el lector puede consultar la sintaxis y descargar ejemplos adicionales sobre aplicaciones específicas en ingeniería del operador ?.

Figura 4.3 Instrucción condicional switch(valor) { case: ... break; default: ... }.

Algunos aspectos importantes a considerar sobre la sentencia condicional **switch()** son los siguientes:



A diferencia de la instrucción **if()**, en la sentencia **switch()** sólo se puede ejecutar el correspondiente **case** si existe una igualdad (desde el punto de vista lógico, es decir valor verdadero o falso) entre la expresión lógica de la instrucción **switch()** y la constante asociado a dicho **case**.



De otra forma, se ejecuta las sentencias relacionadas con la instrucción opcional **default**. Mientras que en la instrucción **if()** no tiene este tipo de restricción, es decir puede evaluar cualquier condición lógica.

Una sentencia switch() no puede tener dos case con constantes iguales o con los mismos valores, es decir case repetidos. Con excepción de sentencias switch() anidadas que puedan tener constantes case iguales (se consideran case locales en la instrucción switch() anidada), es decir una instrucción switch() contenida en otra instrucción switch() donde algún case interno coincida con la misma constante de un case de la instrucción switch() externa.

Por ejemplo, el código 4.6 presenta el siguiente escenario: considere **case 1** asociado a la instrucción **switch(valor)**, este caso es muy diferente al **case 1** que pertenece a la instrucción anidada **switch(valor1)**; observe que ambos casos coinciden con la misma constante, pero el ámbito local marca la diferencia.

El mismo escenario sucede con la sentencia **default**, el cual corresponde a la estructura gramatical de la instrucción **switch()**, a pesar de que se encuentran anidadas dos instrucciones **switch()**, por ejemplo:

switch(expresión lógica_1){switch(expresión lógica_2){...}}

el ámbito local determina el nivel de jerarquía.

Para el caso de sentencias **switch()** con estructura anidadas, cuando la ejecución del programa se realiza en la instrucción **switch** interna o anidada, entonces al encontrar la sentencia correspondiente **break** hace que la ejecución del programa se transfiera hacia la instrucción **switch(**expresión lógica) externa.

En otras palabras, para los niveles de anidamiento de la instrucción **switch**(), siempre será la transferencia en la ejecución del programa del nivel interno donde se ejecuta el sketch hacia el nivel inmediato externo.

Cuando se utilizan constantes de tipo carácter, la fase de compilación las convierte automáticamente a sus equivalentes números enteros, como el caso **case 'a'** que se presenta en el código ejemplo 4.6.

€ Código ejemplo 4.6

```
switch(valor) { case: switch(valor1) { case: ... break; default:... } break; default:... }
switch(valor){//instrucción externa
//Observe que el case 1 coincide con la misma constante de la instrucción
//interna o anidada switch(valor1). Sin embargo, no existen problemas de
//ambigüedad, ya que corresponden a escenarios muy diferentes.
    case 1:// caso diferente al de la instrucción anidada switch(valor1).
       bloque de instrucciones 1;
       break;
    case 'a' ://La fase de compilación sustituye 'a' por constante entera.
       bloque de instrucciones 2;
       switch(valor1){//instrucción anidada.
             case 1://este caso es local a switch(valor1).
                 bloque de instrucciones 3;
              break; //transfiere programa al nivel externo inmediato.
              default:
                 bloque de instrucciones 4;
       }//termina instrucción switch() interna o anidada.
       break;//termina la ejecución de la instrucción switch() externa.
       default:
       bloque de instrucciones 5;
}//termina instrucción switch() externa.
```

La sentencia switch() puede ser sustituida por if()-else-if() para realizar múltiples comprobaciones, pero esta sustitución puede ser engorrosa y generar errores de sintaxis y por lo tanto, no es recomendable. Además, el código resultante no es elegante desde el punto de vista estético en lenguaje C.



Ejemplos con $switch()\{...\}$

Varios ejemplos ilustrativo de la instrucción **switch**(){...}, y también con la versión anidada se tienen preparados en el sitio Web de esta obra.



4.7.3 for $(;;)\{...\}$

La instrucción for genera un lazo cerrado de ciclos de instrucciones, es decir se usa para repetir un número específico de veces un bloque de declaraciones o conjunto de sentencias encerradas entre las llaves delimitadoras {...}. La sintaxis de esta instrucción consta de tres componentes dentro de paréntesis y separadas entre sí por el operador; por ejemplo:

```
for(inicialización; condición de salida; operación aritmética) {//llave que abre.
......
bloque o conjunto de instrucciones;
......
}//llave que cierra.
```

La inicialización es una sentencia de asignación que funciona como variable contador dentro del lazo de la instrucción for.

Cuando la condición de salida es falsa, entonces se finaliza el ciclo de ejecución; el incremento determina la operación aritmética que cambia la variable contador dentro del ciclo o lazo del programa.

El conjunto de sentencias encerradas entre las llaves {...} se ejecuta un número de veces indicado por la variable contador; cuando el programa llega a la llave de cierre } de la instrucción for, entonces se modifica la variable contador por medio de la operación aritmética definida, por lo que se analiza el valor lógico en la condición de salida; si ésta es verdadera, continúa dentro del lazo de programa hasta que la condición de salida alcance el valor falso.

Un aspecto importante de la instrucción for (;;) {...} es que la condición de salida se lleva a cabo siempre al principio del lazo del programa, es decir, el código del bloque de sentencias dentro del lazo de for (;;) {...} no se realizará si la condición de salida es falsa al comienzo.

Por ejemplo, considere el cuadro de código 4.7 donde se genera una onda cuadrada con periodo de 250 mseg usando el puerto 13. Este programa muestra cómo la instrucción **for** se utiliza en aplicaciones de instrumentación electrónica.

El valor inicial de la variable i es cero. La condición de salida es verdadera mientras i sea menor que diez; el tipo de operación aritmética definida para este caso es incrementar la variable de uno en uno. Cuando la variable i=10, entonces la condición de salida es falsa, por lo que el programa

continúa con las instrucciones que estén después de la llave de cierre }.

Código ejemplo 4.7

Instrucción for (;;) {...}

```
for(int i=0; i<10; i++)//declaración interna e inicialización de la variable i.

{ //el lazo de instrucciones se repite diez veces.
    digitalWrite(13, HIGH); //pulso alto en el puerto 13.
    delay(250); //pausa por un 1/4 de segundo.
    digitalWrite(13, LOW); //pulso en bajo en el puerto 13.
    delay(250); //pausa por un 1/4 de segundo.
}
```



La instrucción **for** tiene variantes de programación que proporciona flexibilidad y adaptabilidad en diversas aplicaciones; una variante puede ser que acepta dos o más variables en la sección de inicialización, en este caso se utiliza el operador coma , para separar las variables a inicializar.

Una variante importante de la instrucción for es que permite declarar variables locales dentro de su estructura de programación, por ejemplo: for(float x=78.123, pi=3.1416; i<100; $i++){...}$ y por lo tanto, a las variables x y pi no se requieren declarar como variables globales.

```
También permite declarar e inicializar variables dentro del código asociado a las llaves {...}, por ejemplo: for(float x=78.123, pi=3.1416; i<100; i++){ float y, w=999.89;//declaración de variables en punto flotante. y=x+sin(w*pi); }
```

Otra forma de ilustración es el código en lenguaje C descrito en el cuadro 4.8, observe que las variables **i**, **j**, **k** se declarar e inicializan dentro de la instrucción **for**, la condición de ejecución del código es que la suma de todas las variables sea menor que 200.

Note también que dentro del for se definen diferentes formas de incremento, cada una de ellas

separadas por el operador coma.

€ Código ejemplo 4.8

```
Variante de la instrucción for (;;) {...}
```

```
//Inicializa tres variables i, j, k con diferentes operaciones aritméticas.

for(int i=0, j=15, k=23; i+j+k<200; i++, j=j+3, k=i+j)

{
//el lazo se repite mientras la suma de las tres variables sea menor a 200.

digitalWrite(13, HIGH); //pulso en alto en el puerto 13.

delay(i+j); //pausa por i+j mseg.

digitalWrite(13, LOW); //pulso en bajo en el puerto 13.

delay(k+j); //pausa por (k+j) mseg

}
```

La figura 4.4 muestra el diagrama de flujo de la instrucción for (;;) {...}; contiene la sección de inicialización de las variables contador, condición de salida, bloque de sentencias, tipo de operación aritmética sobre las variables contador.

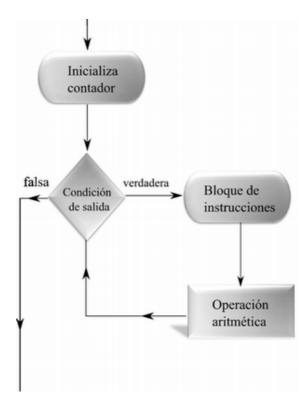


Figura 4.4 Diagrama de flujo de la instrucción for (;;){...}.

Un rasgo distintivo de la instrucción **for** es que no se requiere ocupar todas secciones de esta instrucción, es decir pueden quedar vacías; en otras palabras las secciones de inicialización, condición de salida y operación aritmética son opcionales. Como un caso simple de programación, considere el cuadro 4.9 donde se lee el estado del puerto 13, el **for** se ejecutará mientras el valor lógico de ese pin sea falso o cero. En este caso se encuentran vacías o en blanco las secciones de inicialización y operaciones aritméticas. Como una generalización de este ejemplo, se puede ejecutar un lazo de programa de manera indefinida con el siguiente código: **for**(;;). Para genera retardos por algunos ciclos de reloj es común utilizar la sentencia **for** de la siguiente forma: **for**(**i=0**; **i**<**1000**; **i++**); en este caso no tiene un bloque de instrucciones, debido a que la aplicación de retardo así lo requiere.

Código ejemplo 4.9

```
Campos vacíos de la instrucción for (;;) {...}
```

```
//Secciones de inicialización y operaciones aritméticas en blanco.

for(; i!=1;)

{
//El lazo for se repite mientras el valor del pin 13 sea falso o cero.

i=digitalRead(13); //lee el puerto 13
}
```

La instrucción for admite la forma anidada, es decir una instrucción for dentro de otra instrucción for. En este escenario, la sentencia for externa inicializa una variable contador para ejecutar un número específico de veces al for interno, quien a su vez ejecuta otro número determinado de sentencias. El siguiente ejemplo muestra una aplicación de instrucciones for anidados.



Sintaxis de for (;;)

La instrucción for (; ;) {...} admite declaración de variables locales dentro del ámbito del código que se encuentra delimitado por los operadores {...}. La sintaxis de esta instrucción es diversa y es útil en varias aplicaciones de ingeniería robótica y mecatrónica; ejemplos ilustrativos que explican a detalle la forma utilizar a profundidad la instrucción for se presentan en el sitio Web.

♣ ♣ ♣ Ejemplo 4.7

Desarrollar en lenguaje C el algoritmo para sumar dos matrices $A, B \in \mathbb{R}^{n \times m}$ y presentar el resultado en el monitor serial del ambiente de programación Arduino.

Solución

Sean A y B matrices de dimensión $n \times m$, es decir: $A, B \in \mathbb{R}^{n \times m}$, donde n representa el número de renglones y m el número de columnas. La suma de matrices existe sólo entre matrices de la misma dimensión, $C = (A + B) \in \mathbb{R}^{n \times m}$, se define de la siguiente forma:

$$A + B = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2p} \\ \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{bmatrix} + \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1m} \\ b_{21} & b_{22} & \cdots & b_{2m} \\ \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nm} \end{bmatrix}$$

$$= \begin{bmatrix} a_{11} + b_{11} & a_{12} + b_{12} & \cdots & a_{1m} + b_{1m} \\ a_{21} + b_{21} & a_{22} + b_{22} & \cdots & a_{2m} + b_{2m} \\ \vdots & \ddots & \vdots \\ a_{n1} + b_{n1} & a_{n2} + b_{n2} & \cdots & a_{nm} + b_{nm} \end{bmatrix}$$

El sketch cap4_forfor (disponi-

ble en el sitio Web de esta obra (a)) se encuentra en el cuadro de código Arduino 4.7 con el algoritmo en lenguaje C para realizar la suma de dos matrices A, $B \in \mathbb{R}^{n \times m}$. En la línea 1 se encuentran declarados los pivotes \mathbf{i} , \mathbf{j} como variables globales del tipo constante entero sin signo, utilizados como índices, para referenciar a los elementos de las matrices A y B. Como ejemplo se ha seleccionado matrices cuadradas de dimensión 3×3 de la forma:

$$A = \begin{bmatrix} 5 & 3 & 2 \\ 6 & 4 & 5 \\ 7 & 0 & 1 \end{bmatrix} \qquad B = \begin{bmatrix} -3 & 0 & 1 \\ -10 & 8 & 7 \\ 6 & 3 & 8 \end{bmatrix}$$

De las líneas 4 a la 10 se realiza la declaración de las matrices usando arreglos multidimensionales. Observe que en la línea 4 se inicializa el primer renglón de la matriz A con sus tres columnas. De manera análoga, para los demás renglones, así como los elementos de la matriz B. La velocidad de transmisión serial ha quedado establecida en 9600 Baudios en la línea 12 de la función de configuración setup(). En la línea 14 inicia la rutina letrero encargada de desplegar en forma tabular la información de resultados en el monitor serial Arduino. Note que en la línea 15 se encuentra la declaración de variables locales \mathbf{i} , \mathbf{j} para la manipulación interna del proceso de desplegado de letreros; estas variables son diferentes (a pesar que coinciden en nombre) a la declaración de la línea 1, es decir ocupan diferentes localidades de memoria. La forma de desplegar los letreros en el monitor serial es con la instrucción $\mathbf{for}(\mathbf{j};\mathbf{j})$ en forma anidada; la línea 17 tiene la forma externa \mathbf{j} en la línea 21 la estructura anidada.

El lazo principal de programación loop() inicia en la línea 30, el algoritmo de suma de matrices se

lleva a cabo con instrucciones **for(;;){...}** en forma anidada (línea 31 en forma externa e interna línea 32). El envío de resultados se realiza de la línea 36 a la línea 43; finalmente, el despliegue de resultados en el monitor serial queda detenido por 5 segundos por medio de la función **delay(5000)** ubicada en la línea 44.

La matriz resultante $C \in \mathbb{R}^{3\times 3}$ contiene la suma de las matrices A y B, cuyo procedimiento es ampliamente conocido, el cual se ilustra de la siguiente manera:

$$C = A + B$$

$$= \begin{bmatrix} 5 & 3 & 2 \\ 6 & 4 & 5 \\ 7 & 0 & 1 \end{bmatrix} + \begin{bmatrix} -3 & 0 & 1 \\ -10 & 8 & 7 \\ 6 & 3 & 8 \end{bmatrix}$$

$$= \begin{bmatrix} 5 - 3 & 3 + 0 & 2 + 1 \\ 6 - 10 & 4 + 8 & 5 + 7 \\ 7 + 6 & 0 + 3 & 1 + 8 \end{bmatrix}$$

$$= \begin{bmatrix} 2 & 3 & 3 \\ -4 & 12 & 12 \\ 13 & 3 & 9 \end{bmatrix} \in \mathbb{R}^{3 \times 3}$$

La idea del algoritmo es que los pivotes i y j recorren todas las columnas y matrices de las matrices A y B, por lo que se requiere dos instrucciones **for** en forma anidada.

La figura 4.5 muestra el formato tabular que despliega el sketch cap4_forfor en el monitor serial del ambiente de programación Arduino; la presentación de resultados consiste en exhibir los nueve elementos de cada una de las matrices A, B y la suma se registra en la matriz C, es decir: C = A + B. En el despliegue de datos se indica el número de renglón y de columna, así como informar al usuario durante un periodo de 5 segundos que se volverá a repetir la ejecución del sketch o programa. Para que los resultados del sketch cap4_forfor se desplieguen adecuadamente en el monitor serial del ambiente de programación, es importante maximizar la ventana y activar la opción desplazamiento automático, tal y como se indica en la figura 4.5.



Figura 4.5 Resultado de cap4_forfor.

```
Código Arduino 4.7: sketch cap4_forfor
  Arduino. Aplicaciones en Robótica y Mecatrónica.
                                                            Disponible en Web
  Capítulo 4 Lenguaje C.
  Fernando Reyes Cortés y Jaime Cid Monjaraz.
   Alfaomega Grupo Editor: "Te acerca al conocimiento".
   Sketch cap4_forfor.ino
 1 int i, j; //variables globales, pivotes de las instrucciones for(;;){...}.
 2 const unsigned int renglones= 3, columnas=3; //dimensiones de la matriz.
 3 //declaraciones de matrices A,\ B,\ C \in \mathbb{R}^{\text{renglones} \times \text{columnas}}
 4 float A[renglones][columnas]=\{5,3,2, //\text{elementos}: A[0][0], A[0][1], A[0][2].
                                    6,4,5, //elementos: A[1][0], A[1][1], A[1][2].
                                     7,0,1; //elementos: A[2][0], A[2][1], A[2][2].
 7 float B[renglones][columnas]=\{-3,0,1, //\text{elementos}: B[0][0], B[0][1], B[0][2].
                                    -10.8.7, //elementos: B[1][0], B[1][1], B[1][2].
                                    6,3,8; //elementos: B[2][0], B[2][1], B[2][2].
 9
10 float C[renglones][columnas]; //C = (A + B) \in \mathbb{R}^{\text{renglones} \times \text{columnas}}
11 void setup() { //Rutina de configuración.
        Serial.begin(9600);//Comunicación serie en 9600 Baudios.
12
13 }
14 int letrero(char *s, char *s1, float A[(unsigned int) renglones][(unsigned int)
   columnas]){//rutina para desplegar información en el monitor serial.
       int i, j; //variables locales, pivotes para instrucciones for(;;){...}.
15
        Serial.println(s); //envía letrero al monitor serial.
16
        for(i=0; i < renglones; i++) { //for(; ;) {...} externo.}
17
          Serial.print(s1); //envía segundo letrero al monitor serial.
18
          Serial.print(i); //indica número de renglones.
19
          Serial.print("\t"); //inserta espacio tabular.
20
          for(j=0; j<columnas; j++){ // for(;;){...} interno a anidado.
21
              Serial.print(A[i][j]); //valor de la matriz A[i][j].
22
              Serial.print("\t"); //inserta espacio tabular.
23
24
          }// termina for(; ;){...} anidado.
          Serial.print("\n"); //inserta nueva línea (retorno de carro).
25
        \frac{1}{2}//termina for(; ;){...} externo.
26
       /termina la función letrero(...).
```

```
⊙ Continúa código Arduino 4.7a: sketch cap4_forfor
  Arduino. Aplicaciones en Robótica y Mecatrónica.
                                                       Disponible en Web
  Capítulo 4 Lenguaje C.
  Fernando Reyes Cortés y Jaime Cid Monjaraz.
  Alfaomega Grupo Editor: "Te acerca al conocimiento".
   Continuación del sketch cap4-forfor.ino
28 //rutina principal loop().
29 void loop() {
30 //algoritmo para sumar matrices.
31
       for(i=0; i < renglones; i++) \{ for(;;) \}  externo.
         for(j=0; j; columnas; j++){ //for(;;){...} interno o anidado.
32
             C[i][j]=A[i][j]+B[i][j]; //suma de matrices: C=A+B.
33
          }
34
35
       Serial.println("Elementos de la matriz A"); //envía letrero al monitor serial.
36
                     Col0 Col1 Col2 ", "Ren", A); //envía letrero.
37
       Serial.println("Elementos de la matriz B");
38
                     Col0 Col1 Col2 ", "Ren", B);
       letrero("
39
       Serial.println("Resultado C=A+B");
40
       letrero(" Col0 Col1 Col2", "Ren", C);
41
       Serial.print("Espere 5 segundos."\" n" );
42
       Serial.println("Se repite la ejecución.");
43
       delay(5000); //retardo por 5 segundos.
44
      /consulte la página 61 para ejecutar este sketch.
45 }
```



4.7.4 Instrucción while $()\{...\}$

La instrucción while () pertenece al tipo de sentencias para generar lazos de programas que repetirán un número de veces las sentencias delimitas por las llaves {...} hasta que la condición lógica de salida definida dentro del paréntesis sea falsa (false). Evidentemente el conjunto de instrucciones se encuentran relacionadas para modificar la variable que se emplea en la condición de salida.

La sintaxis general de la instrucción while es la siguiente:

```
while(condición de salida)
{
//mientras la condición de salida sea verdadera, se ejecutará el
  bloque o conjunto de instrucciones;
}
```

Con esta sintaxis la instrucción **while**, la condición de salida se comprueba al principio, dando la posibilidad de que el bloque de instrucciones delimitadas por las llaves no se realice.

Cuando la sentencia a ejecutar consta de una sola instrucción no es necesario delimitarla por paréntesis (sin embargo, se recomienda al lector utilizarlos para mejorar la legibilidad del código y evitar confusiones). Por ejemplo,

```
i=0;
while(i$<=$100) i++;
```

Este código empieza con la inicialización de la variable **i=0**; a continuación la instrucción **while** comprueba que **i** sea menor o igual a 100, la condición lógica es verdadera (true), entonces se ejecuta el incremento sobre la variable **i**, posteriormente se analiza la condición de salida; este proceso se realiza 101 veces, hasta que la condición de salida sea falsa.

Para tener un número indefinido de ciclos de repetición del bloque de sentencias es necesario colocar un valor constante verdadero (true) dentro del paréntesis de la instrucción **while**, por ejemplo: $\mathbf{while}(1) \ \mathbf{i=i+1};$ de esta forma, como la condición de salida siempre es verdadera, se realizará un número infinito de veces el incremento de la variable \mathbf{i} , note que para este caso la condición de salida de la instrucción $\mathbf{while}(1)$, no depende de la manipulación aritmética, ni del procesamiento que se lleva a cabo sobre la variable \mathbf{i} .

Sin embargo, es necesario tener cuidado para estos casos de programación con la instrucción switch(), ya que el incremento de la variable i provocará en un lapso de tiempo un error numérico o desbordamiento debido al tipo de variable que fue definida, por ejemplo del tipo long int podría representar una mejor opción.

La figura 4.6 presenta el diagrama de flujo con la descripción cualitativa del funcionamiento completo de la instrucción while() {...}.

El cuadro de código 4.10 muestra un conjunto de instrucciones ejecutándose 100 veces, hasta que la variable i alcance el valor de 100. Al principio la instrucción while verifica que la condición de salida sea verdadera, si es así, entonces se ejecuta el conjunto de instrucciones. Esto da como posibilidad que si la variable i tiene inicialmente un valor mayor a 100, el conjunto de instrucciones de la instrucción while no se ejecutará.

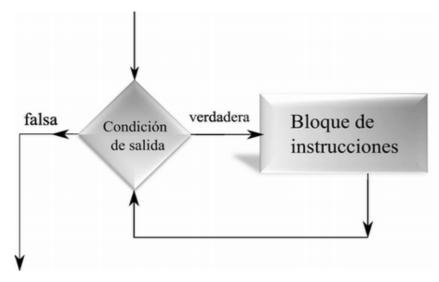


Figura 4.6 Instrucción while $()\{...\}$.

◯ Código ejemplo 4.10

```
Instrucción while() {...}

i=0; //valor inicial de la variable i.

while(i < 100)

{
    //la condición de salida se genera cuando i=100.
    i=i+1; //forma de incrementar la variable i.
    digitalWrite(13, i); //envía dato al puerto pin 13.
    delay(i):
    Serial.println(i);
}
```

4.7.5 Sintaxis do $\{...\}$ while();

La instrucción while también puede tener la siguiente sintaxis: $do\{...\}$ while(); funciona de manera similar a la sintaxis general while() $\{...\}$. La diferencia principal radica en que la condición de salida de la sintaxis $do\{...\}$ while(); se analiza al final del bucle y no al inicio como sucede en while() $\{...\}$, por lo que el lazo $do\{...\}$ while(); siempre se ejecutará al menos una vez.

Observe como parte de la sintaxis el uso del operador ; después del cierre del paréntesis de la palabra reservada while, es decir: do{...}while(...); como ejemplo de la instrucción do{...}while(...); considere el código 4.11: la variable i se inicializa en cero antes de entrar al ciclo de la instrucción dowhile(). Posteriormente, la variable i se incrementa de uno en uno, hasta alcanzar el valor de 99; cuando i=100, la condición de salida i<100 es falsa, terminando la ejecución del ciclo. La figura 4.7 muestra el diagrama de flujo con la descripción cualitativa de la estructura do {...} while (); en sitio Web de esta obra, se abordan ejemplos ilustrativos con la versiones anidadas de las sintaxis: while(){...}; y do{...}while(); y sus aplicaciones en ciencias exactas e ingeniería.

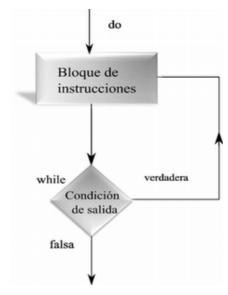


Figura 4.7 do{...}while().

€ Código ejemplo 4.11

Instrucción do{...}while();.

```
i=0;
do
{
// la variable i se incrementa de uno en uno.
i=i+1; //incremento de la variable i.
// el valor actual de la variable i se envía al puerto digital.
digitalWrite(13, i); //envía dato al puerto pin 13.
} while(i<100); //hasta que la variable i alcance el valor de 100.
```

Debe tomarse en cuenta en este ejemplo lo siguiente:

La sintaxis de la instrucción $do\{...\}$ while(...); implica que la variable **i** se incrementa al menos una vez.

La condición de salida i<100 se lleva a cabo después de la llave de cierre que delimita el código de esta instrucción.

En contraste, esto no necesariamente sucede con la sintaxis **while**(i<100){ i=i+1;}, ya que antes de realizar algún tipo de operación aritmética sobre la variable **i**, primero se evalúa la condición de esa variable.

& Ejemplo 4.8

Considere un vector de tiempo t dentro del intervalo de 0 a 5 segundos con incrementos de una décima de segundo; obtener el cálculo de la función sen(t) con la sintaxis de la instrucción while () $\{...\}$.

Solución

El sketch **cap4_while** descrito en el cuadro Arduino 4.8 contiene el código fuente en lenguaje C para obtener el cálculo de la función trigonométrica sen(t) para el intervalo de tiempo $t \in [0, 0.1, 02, \cdots, 4.9, 5]$ segundos. Para este ejemplo, se considera la evolución en el tiempo como múltiplos de 0.1, es decir corresponde a tiempo discreto $t_k = kh$, donde $h = 0.1 \in \mathbb{R}_+$ y k es un número entero positivo, que se incrementa de uno en uno.

Código Arduino 4.8: sketch cap4_while Arduino. Aplicaciones en Robótica y Mecatrónica. Disponible en Web Capítulo 4 Lenguaje C. Fernando Reyes Cortés y Jaime Cid Monjaraz. Alfaomega Grupo Editor: "Te acerca al conocimiento". Sketch cap4_while.ino 1 float h=0.1; //periodo de muestreo de una décima de segundo. 2 float t; /*vector tiempo.*/ 3 void setup() { //Rutina de configuración. Serial.begin(9600); //Comunicación serie en 9600 Baudios. 5 } 6 void loop() { //rutina principal loop() t=0; // vector tiempo inicia en 0 segundos Serial.println("Sketch cap4_while"); while (t<5.0){ //mientras el vector tiempo sea menor a 5 segundos. 9 Serial.print("Valor de sen(t)="); //letrero. 10 Serial.print("\t"); /*inserta espacio tabular en el monitor serial.*/ 11 Serial.println(sin(t)); /*envía información al monitor serial.*/ **12** t=t+h; /*incremento del vector tiempo en décimas de segundo.*/ 13 delay(1); /*retardo requerido de un milisegundo para visualizar resultados.*/ 14 15 16 }

En la línea 1 se declara \mathbf{h} como variable de tipo flotante para incrementos de tiempo por décimas de segundo, así como la variable \mathbf{t} que lleva el registro del vector tiempo (línea 2).

La comunicación serial USB queda establecida en 9600 Baudios (línea 4), la rutina principal **loop()** inicia a partir de la línea 4. El valor inicial del vector tiempo queda definido en cero segundo como se muestra en la línea 7.

La instrucción while () $\{...\}$ inicia en la línea 9, lo primero que realiza es verificar la condición de salida t < 5.0, mientras tenga un valor lógico verdadero se ejecuta el código de programación contenido dentro de los paréntesis $\{...\}$.

En las líneas 10 y 11 se envía al monitor serial del ambiente de programación Arduino (a la computadora por USB) un letrero informativo y espacio tabular, respectivamente. En la línea 12 se transmite el cálculo de la función senoidal sen(t) para un tiempo t, el cual evoluciona con incrementos de una décima de segundo como se indica en la línea 13.

Con la finalidad de que el usuario visualice los resultados en el monitor serial del ambiente de programación Arduino, en la línea 14 se genera un retardo de un milisegundo entre la información transmitida desde la tarjeta Arduino a la computadora.

Note que para ejecutar el código contenido entre las llaves {...} de la instrucción while, primero se verifica que el valor lógico de la condición de salida while(t<5.0) sea verdadero (línea 9), de ahí que es necesario asegurar que el valor inicial de t sea cero (línea 7).

También observe que no es suficiente inicializar $\mathbf{t}=\mathbf{0}$ en la línea 2, ya que esto funcionaría sólo la primera vez (con excepción de que eso sea la intención), la estructura de la instrucción **while** realiza los incrementos correspondientes y cuando la condición $\mathbf{t}<\mathbf{5.0}$ sea falsa, es decir para $\mathbf{t}\geq\mathbf{5}$ el código contenido en las llaves $\{...\}$ no se ejecutará.

Recuerde que la rutina principal **loop** repetirá indefinidamente este ciclo, hasta que el usuario descargue a la tarjeta otro sketch.

Para descargar y ejecutar el código del sketch **cap4_while** en las tarjetas Arduino y visualizar los resultados en el monitor serial del ambiente de programación Arduino **consulte la página 61**.



j=i+200;

4.7.6 Sentencia break

La sentencia **break** se emplea para finalizar un **case** dentro de una instrucción **switch(){...}** (ver subsección 4.7.2) y también para finalizar los ciclos de ejecución de las instrucciones **for(;;){...}**, **while(){...}** y **do{...}while()**;. Considere los códigos 4.12 y 4.13 donde se emplea la sentencia **break** dentro de las instrucciones **for** y **while**, respectivamente; si la variable **i** toma el valor de 88, se finaliza el ciclo de las instrucciones **for** y **while**, continuado el programa con la sentencia j=i+200;.

€ Código ejemplo 4.12

```
Uso de la sentencia break en la instrucción for
int i,j;
for(i=0; i<100; i++)
{
    if (i==88) break;//salta al código: j=i+200;
}
j=i+200;
```

Código ejemplo 4.13

```
Uso de la sentencia break en la instrucción while
int i=0, j;
do{
    if (i==88) break;//salta al código: j=i+200;
    i=i+1;
}while(i<100);
```

Generalmente, cuando la sentencia **break** se emplea con las instrucciones de bucle o lazo de programas como **for(;;)**, **do{...}while()**; y **while()**{...} su uso está asociado con un tipo de instrucción condicional, por ejemplo: **if()**{...}.

4.7.7 Sentencia continue

La sentencia **continue** se puede utilizar con instrucciones de lazo de programa como **for, while** y **do-while**; **continue** no termina el ciclo de programa de esas instrucciones como sucede con **break**, más bien, genera una nueva iteración del lazo del programa como se muestra en los códigos 4.14 y 4.15:

€ Código ejemplo 4.14

Uso de la sentencia continue en la instrucción do-while

```
int i=0, j;
do{
    if (i==88) continue;//continúa dentro del bucle do-while.
    i=i+1;
}while(i<100);
j=i+200;
```

€ Código ejemplo 4.15

Uso de la sentencia continue en la instrucción for

```
int i=0, j;
for (i=0; i<100; i++){
    if (i==88) continue;//continúa dentro del lazo de programa for.
}
```

En las instrucciones **do-while** y **while**, la sentencia **continue** hace que el control del programa analice el estado lógico de la condición de salida, si es verdadera prosigue dentro del código de instrucciones del bucle o lazo de programa.

Por otro lado, para la instrucción **for** cuando se ejecuta la sentencia **continue** primero se incrementa la variable contador, después se verifica la condición de salida, en el caso que sea verdadera sigue con el bloque de instrucciones dentro del bucle o lazo de control.

Ejemplos adicionales

Para mejorar la comprensión del lenguaje C, ejemplos adicionales de las instrucciones condicionales if(){...}, if(){...} else{...}, switch(){...} y de lazo for(;;) {...}, while(){...} y do{...}while() con sus versiones anidadas se encuentran disponibles en el sitio Web de esta obra.

De la misma forma, también se encuentran disponibles ejemplos de las sentencias **break** y **continue**, así como una sección destinada a la sentencia **goto**.

4.8 Resumen 121

4.8 Resumen



E lenguaje de programación C se ha convertido en una importante herramienta de desarrollo para controlar y automatizar experimentos de las ciencias exactas, así como una diversidad de aplicaciones de ingeniería. Particularmente, en robótica y mecatrónica la implementación práctica de algoritmos de control se realiza en su totalidad en lenguaje C, adquisición de datos (posición y velocidad del robot), manejo de puertos, envío de comandos o par aplicado al servoamplificador, comunicación con la computadora para desplegar información gráfica, mientras el sketch se está ejecutando en la tarjeta Arduino dedicado exclusivamente al control del sistema.

Otra ventaja de este lenguaje es que en la actualidad la gran mayoría de microcontroladores se programan en C, por lo que la implementación práctica de los algoritmos es mucho más sencilla. Dentro de este escenario, el sistema Arduino presenta un ambiente de programación con editor y compilador integrado del lenguaje C con las herramientas necesarias para depurar los errores de sintaxis del sketch o programa, descargar y ejecutar el sketch y visualizar la información requerida por el usuario en el monitor serial del propio ambiente.

La sintaxis del lenguaje C se refiere a las reglas gramaticales para escribir correctamente las instrucciones y sentencias, uso de operadores y modificadores para manipular computacionalmente tipos de datos a nivel byte y bits. C como lenguaje de programación, contiene como elementos de programación identificadores para declarar y proporcionar nombres a variables y funciones, tipos de datos con números enteros de uno, dos y cuatro bytes, números reales (flotantes y con doble precisión), operadores aritméticos, lógicos, relacionales e instrucciones que admiten varias estructuras de programación como: if() {...} else{...}, switch() {...}, do{...} while() y for(;;) {...}.

Además, dentro de los potenciales que tiene el lenguaje C se encuentran las funciones, también conocidas como subrutinas o procedimientos, que representan un bloque de instrucciones que realizan una actividad específica y ayudan a ser más simple el código de un programa.

4.9 Referencias selectas



E lenguaje C es una potente herramienta que permite implementar aplicaciones de robótica, mecatrónica y en general de cualquier área de la ingeniería y ciencias exactas. Existe una vasta literatura sobre programación en lenguaje C; a continuación se recomienda literatura específica para profundizar en lenguaje C.

web http://arduino.cc/en/Reference/HomePage

Un clásico de la literatura del lenguaje C es:



Brian Kernighan and Dennis Ritchie. "The C Programming Language". Prentice-Hall, 1978.

Otras obras importantes del lenguaje C son las siguientes:



Herber Schildt. "C: The Complete Reference". 4th Edition, McGraw-Hill Osborne Media, 2000.

Herber Schildt. "C: Manual de referencia". Cuarta edición, McGraw-Hill, 2001.

Stephen G. Kochan. "Programming in C: A complete introduction to the C language". Third Edition, Sams Publishing, 2004.



4.10 Problemas propuestos

 \P N esta sección se presentan una serie de ejercicios con el propósito de mejorar los conocimientos dadquiridos por el usuario sobre aspectos de sintaxis e implementación de algoritmos prácticos y su ejecución en las tarjetas Arduino.

- 4.10.1 Considere a x, y variables de tipo entero (int) y el uso del operador ++ como: y=++x;. Desde el punto de vista de la gramática del lenguaje C las siguientes sentencias son correctas:
 - a) y=+(+x);
 - b) y=(++x);
 - c) y=(++)x;

Diga si esas sentencias son equivalentes (producen el mismo resultado) a la sentencia y=++x;

4.10.2 Suponga que tiene la siguiente declaración de variables enteras:

int y, x=50;

Sea la siguiente declaración:

y=-+x;

¿Cuál es el resultado correcto y qué valor final tendrá la variable \mathbf{x} ?

- a) y=50;
- b) y=51;
- c) y=-50;
- d) y=-51;

¿Cuál sería el código equivalente a y=-+x;?

4.10.3 Considere que las variables x, y son del tipo entero. Es decir int y, x=50; ¿Es correcta la sintaxis de las siguientes sentencias?

- a) y=x-+;
- b) y=x+-;

Argumente su respuesta.

4.10.4 Diseñe un algoritmo para encontrar los valores mínimo y máximo de las siguientes funciones:

- a) sen(x).
- b) $\cos(x)$.
- c) x^2 .
- d) $\frac{2}{11}x^{35}$.
- e) tanh(x).
- f) atan (x).

Considere el intervalo $x \in [-5, 5]$. El resultado de mínimo y máximo desplegarlo en el monitor serial del ambiente de programación Arduino.

4.10.5 Realizar el algoritmo para aproximar la derivada de las siguientes funciones f(x):

- a) sen(x).
- b) $\cos(x)$.
- c) x^2 .
- d) $\frac{2}{11}x^{35}$.

- e) tanh(x).
- f) atan (x).

Sugerencia: utilice el método de Euler, sea f(x) una función continua en x, entonces: $\dot{f}(x) \simeq \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}$. Los algoritmos respectivos, ejecutarlos en algún modelo de tarjeta Arduino y desplegar el resultado en el monitor serial del ambiente de programación. Comparar los resultados obtenidos con las correspondientes expresiones analíticas de las derivadas de esas funciones en MATLAB (también, visualizar los resultados en forma de texto, para una fácil comparación).

4.10.6 Calcular la integral numérica de:

- a) $\int_0^5 \sin(x) dx$.
- b) $\int_0^5 \cos(x) dx.$
- c) $\int_0^5 x^2 dx$
- d) $\int_0^5 \frac{2}{11} x^{35} dx$
- e) $\int_0^5 \tanh(x) dx$
- f) $\int_0^5 \arctan(x) dx$

Sugerencia



Para obtener el algoritmo discreto de una integral, se utiliza el método numérico trapezoidal (también conocido como método discreto de integración de Euler), el cual está dato por la siguiente expresión:

$$Int_k = Int_{k-1} + hf(x_k)$$

donde Int_k es el valor de la integral en la k-ésima iteración, Int_{k-1} es el valor discreto de la integral en la iteración anterior, $f(x) \in \mathbb{R}$ es la función a integrar, $x \in \mathbb{R}$ es el argumento, y $h \in \mathbb{R}_+$ es el paso de integración.



El algoritmo resultante ejecutarlo en algún modelo de tarjeta Arduino y desplegar el resultado en el monitor serial del ambiente de programación.

Comparar los resultados obtenidos con la evaluación numérica de las correspondientes funciones en MATLAB (es preferible visualizar los resultados en forma de texto para llevar a cabo dicha comparación).

Apuntadores, estructuras y uniones

struct robot{
 float posiciones[6];
 float velocidades[6];
 float torques[6];
 int puerto0 : 1;
 int puerto1 : 1;
} robotA, robotB, robotC;

Capítulo Web

- 5.1 Introducción
- 5.2 Apuntadores
- 5.3 Estructuras
- 5.4 Uniones
- 5.5 Resumen
- 5.6 Referencias selectas
- 5.7 Problemas propuestos

Competencias

Presentar los elementos claves de programación de apuntadores, estructura de datos y uniones a través de ejemplos didácticos y pedagógicos que se ejecutan en las plataformas electrónicas de tarjetas Arduino, con el propósito de implementar aplicaciones en ciencias exactas e ingeniería.

Desarrollar habilidades en:



Programación de aplicaciones en ingeniería y ciencias exactas a través de apuntadores, estructura de datos y uniones.



Manejo de variables puntero.

Inicialización y asignación dinámica de memoria.

Acceso y manipulación de los campos de estructura de datos.

Acceso y manipulación de campos de unión de datos.



Librerías y funciones Arduino

Capítulo

```
#include <stdlib.h>
#include <Ethernet.h>
#include <Servo.h>
void setup(){
    Serial.begin(9600);
}
void loop(){
    robot_control();
}
```

- 6.1 Introducción
- 6.2 Librerías Arduino
- 6.3 Funciones Arduino
- 6.4 Resumen
- 6.5 Referencias selectas
- 6.6 Problemas propuestos

Competencias

Presentar las librerías y funciones Arduino para aprovechar los recursos y características de la plataforma electrónica del conjunto de modelos de tarjetas, con la finalidad de obtener mejor eficiencia y desempeño de programación en aplicaciones de ingeniería y ciencias exactas.

Desarrollar habilidades en:



Programación con librerías estándar del lenguaje C.



Eficiencia y desempeño con librerías Arduino.

Aplicaciones de funciones Arduino.

6.1 Introducción 129

6.1 Introducción



Las librerías o bibliotecas del lenguaje C son una colección de funciones que contienen el código objeto de las funciones proporcionadas con el compilador y se encuentran agrupadas por razones de eficiencia o restricciones para realizar actividades de manejo de puertos digitales entrada/salida y dispositivos electrónicos, cadenas, matemáticas, manipulación y procesamiento de datos, asignación dinámica de memoria, y uso de recursos de los microcontroladores ATMEL.

El estándar ANSI define propiedades, tamaño y contenido para una biblioteca y la gran mayoría de estas funciones han sido rediseñadas para el sistema Arduino debido a que se carece de un sistema operativo y los recursos de las tarjetas electrónicas son limitados en comparación con una computadora, ya que debe recordarse que la filosofía Arduino se encuentra orientada a los sistemas empotrados o (embedded systems), por tal motivo las funciones requieren una adecuación para ser transportables.

Por otro lado, debido a las características específicas de los microcontroladores de la familia ATMEL, se requieren realizar funciones especializadas para aprovechar los recursos de hardware y software de sus plataformas electrónicas, por lo que también se cuentan con librerías Arduino.

Las librerías son archivos que contienen el código objeto y nombre de cada una de las funciones, más la información de reubicación en memoria y enlaces de compilación para las tarjetas Arduino. Cuando un sketch hace referencia a una función, entonces el compilador toma el código correspondiente a esta función contenida en la librería y añade sólo el código objeto de esa función al sketch, de esta forma de la gran cantidad de funciones que contiene una librería, sólo se añade el código de las funciones que se utilizan en el sketch.

Las funciones de las librerías estándar del lenguaje C, así como las del sistema Arduino trabajan con sus propios tipos de datos y variables a las que un sketch debe acceder, por tal motivo, adicionalmente se requiere de un archivo cabecera o header donde se define la sintaxis de la función, es decir, el tipo de variables que requiere como argumentos de entrada y el tipo de datos que retorna; los archivos cabecera se colocan al inicio de un sketch por medio de la directiva #include <archivo.h>, donde la extensión de este archivo proviene de header.

Por ejemplo, para utilizar funciones matemáticas se emplea **#include**<**math.h**>, para control de servomotores se utiliza **#include**<**Servos.h**>, y para el manejo y manipulación de cadenas **#include**<**string.h**>.

Existe una variedad de librerías estándar del lenguaje C y del sistema Arduino, sin embargo, en

este capítulo sólo presentan las funciones que se han utilizado en la presenta obra. No obstante, se proporcionan las direcciones donde el usuario puede encontrar documentación en información detallada de todas las librerías.



6.2 Librerías Arduino

En N esta sección se describen las funciones básicas del sistema Arduino que permiten programar puertos digitales, lectura de canales analógicos y en general tener acceso a los recursos de la tarjeta electrónica.

El ambiente de programación Arduino, como otras plataformas de programación, puede ser extendido a través del uso de librerías, las cuales proveen funcionalidad extra a los sketches, por ejemplo para trabajar con hardware o manipulación y procesamiento de datos.

Las librerías avr-libc son un subconjunto de las librerías estándar del lenguaje C y contiene las funciones específicas para utilizar los recursos de los microcontroladores RISC ATMEL AVR, proporcionan la forma de trabajo para realizar un amplio espectro de aplicaciones en ciencias exactas e ingeniería. La tabla 6.1 muestra tan sólo algunas de las librerías del lenguaje C y del sistema Arduino.

El siguiente enlace electrónico contiene la documentación e información detallada sobre todas las librerías estándar del lenguaje C y del sistema Arduino:

http://www.nongnu.org/avr-libc/user-manual/modules.html



Librerías y funciones Arduino

Diversos ejemplos ilustrativos con funciones y librerías Arduino con aplicaciones para ingeniería y ciencias exactas se encuentran presentes en el sitio Web de este libro.

Tabla 6.1 Librerías del lenguaje C y del sistema Arduino.

| Librería | Descripción |
|--|--|
| #include <allocate.h></allocate.h> | Asignación temporal de memoria stack. |
| #include <math.h></math.h> | Funciones matemáticas. |
| #include <stdio.h></stdio.h> | Funciones estándar de entrada/salida. |
| #include <stdlib.h></stdlib.h> | Librerías estándar. |
| #include <string.h></string.h> | Manejo y procesamiento de cadenas. |
| #include <stdint.h></stdint.h> | Tipos estándar de enteros. |
| #include <avr boot.h=""></avr> | Soporte y utilidades del arrancador/cargador (bootloader). |
| #include <avr cpufunc.h=""></avr> | Funciones especiales del microcontrolador AVR. |
| #include <avr eeprom.h=""></avr> | Manejo de memoria EEPROM. |
| #include <avr interrupt.h=""></avr> | Manejo de interrupciones. |
| #include <avr io.h=""></avr> | Manejo y definiciones de dispositivos entrada/salida. |
| #include <ethernet.h></ethernet.h> | Librerías para comunicación por medio de Ethernet. |
| #include <arduinorobot.h></arduinorobot.h> | Librerías para el control de la tarjeta Robot. |
| #include <gsm.h></gsm.h> | Conexión a red GSM/GRPS con GSM shield. |
| #include <liquidcrystal.h></liquidcrystal.h> | Control del exhibidor de cristal líquido. |
| #include <sd.h></sd.h> | Lectura y escritura de tarjetas SD. |
| #include <servo.h></servo.h> | Funciones para manejo y control de servomotores. |
| #include <spi.h></spi.h> | Comunicación con dispositivos usando el bus de Interface serial periférica (SPI). |
| #include <stepper.h></stepper.h> | Control de motores a paso. |
| #include <wifi.h></wifi.h> | Comunicación por medio de Internet usando Arduino WiFi shield. |
| #include <wire.h></wire.h> | Comunicación para transmitir/recibir datos por medio del protocolo TWI/I2C. |



6.2.1 Librerías stdio.h

La librería **stdio.h** es un archivo cabecera que se coloca en las primeras líneas del sketch a través de la siguiente directiva : **#include** <**stdio.h**>, el cual contiene la sintaxis y declaración de funciones estándar entrada/salida (IO), que se emplea en avr-libc. Debido a que las tarjetas electrónicas Arduino no contienen un sistema operativo y sus características de hardware están orientadas a la de un sistema empotrado o (*embedded system*), sólo se implementa un conjunto limitado de funciones estándar IO.



6.2.2 Librerías stdlib.h

El archivo cabecera **stdlib.h** se emplea en el *header* (primeras líneas) con la directiva **#include** <**stdlib.h**>, el cual contiene la sintaxis de diversas librerías estándar del lenguaje C, entre las que se encuentran funciones que convierten cadena de caracteres a números enteros, flotantes (y viceversa), cómputo del valor absoluto de números enteros, generación de números aleatorios.

Particularmente, en esta subsección presentamos sólo la descripción de algunas funciones que han sido de utilidad para las finalidades de la presente obra, tales como las funciones de asignación/liberación dinámica de memoria para las tarjetas Arduino.

void * malloc(num_bytes)

Esta función asigna memoria en forma dinámica con una longitud de número de bytes definido por el argumento de entrada **num_bytes**. La asignación de memoria depende de la disponibilidad y características de memoria que tenga la tarjeta electrónica. Retorna un apuntador de tipo void direccionando el inicio de la localidad de memoria asignada. Si la tarjeta no tiene suficiente memoria para asignar, entonces retorna un apuntador nulo (NULL).

Una forma ampliamente utilizada para la función **malloc(...)** es la inicialización de apuntadores y por medio de un **casting** se puede acoplar adecuadamente el apuntador que retorna a cualquier tipo de variable apuntador. El código ejemplo 6.1 muestra la forma como debe emplearse la función **malloc(...)** en apuntadores de tipo flotante, char y entero.

free(void *apuntador)

La función **free(...)** libera la memoria asignada dinámicamente al **apuntador** por la función **malloc(...)**. Si **apuntador** es nulo, ninguna acción ocurre.

Mayor información sobre las funciones de la librería stdlib.h como asignación dinámica de memoria y otras más se pueden consultar en:

http://www.nongnu.org/avr-libc/user-manual/group_avr_stdlib.html

El código ejemplo 6.1 muestra la forma que debe utilizarse la función **free(...)** en apuntadores de tipo flotante, char y entero.

Observe que en la primera línea o en la cabecera del sketch se utiliza la directiva **#include** <**stdlib.h**>, necesaria para utilizar las funciones de asignación dinámica de memoria **malloc(...)** y para liberarla **free(...)**.

Para efectos ilustrativos sobre el manejo de asignación dinámica de memoria, se declaran tres tipos diferentes de apuntadores (float *z, char *c e int *i), en esta declaración los apuntadores no están inicializados, es decir, no apuntan a una dirección de memoria específica, por lo que no conviene su empleo en estas circunstancias.

Una forma de inicializar a los apuntadores es por medio de asignación de memoria a través de la función malloc(...), por ejemplo para asignar los cuatro bytes que requiere una variable de tipo flotante se utiliza la función sizeof(z), la cual calcula el número de bytes que necesita el apuntador de tipo flotante; esta función retorna 4 bytes que sirven como argumento de entrada a la función malloc(...), es decir: malloc(sizeof(z)); esta función regresa un apuntador de tipo void direccionando a la localidad de memoria asignada.

Sin embargo, \mathbf{z} es un apuntador de tipo flotante, la forma de poder acoplar el resultado retornado es por medio de un casting, es decir:

z=(float *)malloc(sizeof(z));//convierte el apuntador de tipo void a flotante.

En forma parecida se realiza la asignación de memoria para los casos de apuntadores tipo **char** e **int**:

c=(char *)malloc(sizeof(c));//convierte el apuntador de tipo void a char.

i=(int *)malloc(sizeof(i));//convierte el apuntador de tipo void a int.

Para liberar la cantidad de memoria asignada al apuntador flotante z, se utiliza la función free(z), de manera análoga, para los casos de los apuntadores de tipo char e int se utiliza free(c) y free(i), respectivamente.

€ Código ejemplo 6.1

Sketch para asignación dinámica de memoria en apuntadores.

```
#include <stdlib.h>//librería para las funciones malloc(...) y free(...).
float *z;//declaración de un apuntador tipo flotante.
char *c;//declaración de un apuntador tipo char.
int *i;//declaración de un apuntador tipo entero.
void setup(){//subrutina de configuración.
    Serial.begin(9600); //configuración de comunicación serial.
void loop(){//Inicia lazo principal de programación del sketch.
    Serial.print("Asignación dinámica de memoria para apuntadores");
    z=(float *)malloc(sizeof(z));//retorna apuntador tipo punto flotante.
    c=(char *)malloc(sizeof(c));//retorna apuntador tipo char.
    i=(int *)malloc(sizeof(i));//retorna apuntador tipo entero.
    *z=M_PI;//valor asignado en la localidad de memoria que apunta z.
    *c='n';//valor asignado en la localidad de memoria que apunta c.
    *i=4509;//valor asignado en la localidad de memoria que apunta i.
    Serial.print("Valor asignado al apuntador tipo flotante= ");
    Serial.println(*z,5);//desplegar información con 5 fracciones.
    Serial.print("Valor asignado al apuntador tipo char=");
    Serial.println(*c);//desplegar información del apuntador tipo char.
    Serial.print("Valor asignado al apuntador tipo entero= ");
    Serial.print(*i);//desplegar información del apuntador tipo entero.
    free(z);//libera espacio de memoria (4 bytes) asignado al apuntador z.
    free(c);//libera espacio de memoria (1 byte) asignado al apuntador {\bf c}.
    free(i);//libera espacio de memoria (2 bytes) asignado al apuntador i.
    Serial.println("En 3 segundos se ejecuta una vez más el sketch.\n\n'");
    delay(3000);//retardo en la transmisión serial.
```

Note el uso de la función sizeof(...) para calcular el número de bytes de un tipo de variable y asignar la cantidad de memoria adecuada. Otros ejemplos ilustrativos con las funciones malloc(...) y free(...) se encuentran en el capítulo 5 Apuntadores, estructuras y uniones.

Alfaomega Arduino. Aplicaciones en Robótica y Mecatrónica

Fernando Reyes Cortés • Jaime Cid Monjaraz



6.2.3 Funciones matemáticas

La librería de funciones matemáticas se denomina **math.h**, y se inserta en las primeras líneas del header (cabecera del sketch o programa) a través de la siguiente directiva: **#include** <**math.h**>. Incluye un conjunto de constantes útiles en ingeniería y ciencias exactas relacionadas con los números π , e, y $\sqrt{2}$; también, tiene varias funciones trigonométricas, hiperbólicas, exponenciales, valor absoluto, raíz cuadrada, entre otras más. Estas funciones, generalmente aceptan argumentos doble flotante (**double**) y retornan un valor del tipo **double**.

Las tablas 6.2 y 6.3 contienen las constantes y funciones matemáticas, respectivamente.

Tabla 6.2 Constantes matemáticas #include <math.h>.

| Constante | Descripción |
|---|----------------------|
| #define M_PI 3.14159265358979323846 | Número π |
| #define M_PI_2 1.57079632679489661923 | $\frac{\pi}{2}$ |
| #define M_PI_4 0.78539816339744830962 | $\frac{\pi}{4}$ |
| #define M_1_PI 0.31830988618379067154 | $\frac{1}{\pi}$ |
| #define M_2_PI 0.63661977236758134308 | $\frac{2}{\pi}$ |
| #define M_E 2.7182818284590452354 | Número e |
| #define M_LOG2E 1.4426950408889634074 | $\log_2(e)$ |
| #define M_LOG10E 0.43429448190325182765 | $\log_{10}(e)$ |
| #define M _ LN2 0.69314718055994530942 | $\log_e(2)$ |
| #define M_LN10 2.30258509299404568402 | $\log_e(10)$ |
| #define M_2_SQRTPI 1.12837916709551257390 | $2\sqrt{\pi}$ |
| #define M_SQRT2 1.41421356237309504880 | $\sqrt{2}$ |
| #define M_SQRT1_2 0.70710678118654752440 | $\frac{1}{\sqrt{2}}$ |
| #define INFINITYbuiltin_inf() | Infinito ∞ |
| #define NAN _builtin_nan("") | Infinito ∞ |

Tabla 6.3 Funciones matemáticas #include <math.h>.

| Función | Descripción |
|----------------------------------|---|
| double cos(double x) | Función trigonométrica coseno, retorna un valor en $[-1, 1]$. |
| double sin(double x) | Función trigonométrica seno, retorna un valor en $[-1, 1]$. |
| double tan(double x) | Función trigonométrica tangente, retorna un valor entre $[-\infty, \infty]$, si $x \in [-\infty, \infty]$. |
| double fabs(double x) | Valor absoluto $ x , x \in \mathbb{R}$. |
| double fmod(double x, double y) | Retorna el residuo de la división $\frac{x}{y}$. |
| double sqrt(double x) | Retorna la raíz cuadrada (no negativa): $\sqrt{x}, x \in \mathbb{R}_+$. |
| double square(double x) | Retorna el cuadrado $x^2, x \in \mathbb{R}$. |
| double exp(double x) | Función exponencial e^x , $x \in \mathbb{R}$. |
| double pow(double x) | Función exponencial calcula $x^y, x, y \in \mathbb{R}$. |
| double cosh(double x) | Función coseno hiperbólico $x \in \mathbb{R}$. |
| double sinh(double x) | Función seno hiperbólico $x \in \mathbb{R}$. |
| double tanh(double x) | Función tangente hiperbólico $x \in \mathbb{R}$, retorna un valor en $[-1,\ 1].$ |
| double acos(double x) | Función arco coseno $x \in \mathbb{R}$; retorna un valor en el intervalo $[0, \pi]$ radianes. Ocurre un error para argumentos que no están en $x \in [-1, 1]$. |
| double asin(double x) | Función arco seno $x \in \mathbb{R}$; retorna un valor comprendido en el intervalo $\left[-\frac{\pi}{2},\ \frac{\pi}{2}\right]$ radianes. Ocurre un error para argumentos que no están en $x \in [-1,1]$. |
| double atan(double x) | Función arco tangente $x \in \mathbb{R}$; retorna un valor comprendido en el intervalo $\left[-\frac{\pi}{2},\ \frac{\pi}{2}\right]$ radianes. |
| double atan2(double y, double x) | Función arco tangente de $\frac{y}{x}$ con $x, y \in \mathbb{R}$; toma en cuenta los signos de x y y para determinar el cuadrante del valor que retorna, el cual se ubica en el intervalo $\left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$ radianes. |
| double log(double x) | Calcula el logaritmo natural de $x, \in \mathbb{R}_+$. |
| double round(double x) | Redondea al entero más cercano de $x \in \mathbb{R}$. |

La tabla 6.4 muestra el conjunto de constantes (variables predefinidas) del sistema Arduino, tales como: booleanas (**true** y **false**), estados lógicos (**LOW** y **HIGH**), configuración de puertos **INPUT, INPUT_PULL** y **OUTPUT** que se utilizan como argumentos de entrada en diversas funciones que programan los recursos de la plataforma de las tarjetas Arduino (**no requieren de un archivo cabecera**).

Tabla 6.4 Constantes empleadas en el sistema Arduino.

| Constante | Descripción |
|--------------|--|
| INPUT | Se utiliza como argumento o pase de parámetro en la función pinmode() para configurar el modo entrada (alta impedancia) en los puertos digitales. |
| OUTPUT | Se emplea como argumento de entrada en la función pinmode() para configurar los puertos digitales como salida (baja impedancia). |
| INPUT-PULLUP | Los microcontroladores ATMEGA que emplean las tarjetas Arduino contienen resistencias pullup que se encuentran conectadas de manera interna a la fuente interna de alimentación, y que pueden acceder mediante programación usando INPUT_PULLUP como argumento de entrada en la función pinmode(). |
| true | Para definir el valor Booleano verdadero o 1 se utiliza true. |
| false | La constante false define el valor Booleano cero o falso. |
| LOW | El significado de LOW depende de la forma en que se encuentren programados los puertos digitales, por ejemplo cuando un puerto funciona como entrada mediante pinmode(), entonces una lectura a través de digitalRead(), retronará un nivel LOW si existe un voltaje de 2 Volts o menor en dicho puerto. Cuando el puerto está configurado como salida con pinmode() y se escribe un nivel LOW en el puerto con digitalWrite(), entonces este pin adquiere 0 volts. |
| HIGH | La interpretación de HIGH depende de si el puerto digital está programado como entrada o salida (en ambos casos configurado con la función $pinmode()$). Cuando un puerto se encuentra como entrada, una lectura en este puerto usando $digitalRead()$ retornará un valor lógico HIGH, si el voltaje en dicho puerto es mayor o igual a 3 volts. Un puerto digital puede estar configurado como entrada y subsecuentemente habilitado en HIGH con la función $digitalWrite$, entonces se activarán la resistencias internas de pullup de $20~\mathrm{K}\Omega$, la lectura de dicho puerto será HIGH a menos que tenga una resistencia de pullup en LOW por la circuitería externa. Cuando el puerto digital está configurado como salida y es puesto en HIGH usando $digitalWrtie()$, significa que dicho pin tendrá un valor de $3.3~\mathrm{V}$ a $5\mathrm{V}$. |

Observe que la constantes **true** y **false** se escriben con letras minúsculas, mientras que **HIGH**, **LOW**, **INPUT_INPUT_PULLUP** y **OUTPUT** se encuentran en mayúsculas.



6.3 Funciones Arduino

En la sección se describen las funciones fundamentales del sistema Arduino que permiten programar los recursos de la plataforma electrónica de las tarjetas Arduino, como por ejemplo: configurar puertos digitales entrada/salida, lectura de canales analógicos, velocidad de comunicación serial USB, envío de información al monitor serial, enlace y comunicación con ambientes de programación como MATLAB, LabView, etc.

Estas funciones no requieren de un archivo cabecera o *header*, ni la directiva **#include** <archivo.h>.



6.3.1 Funciones fundamentales

Las funciones de configuración **setup**() y la del lazo principal del sketch **loop**() representan la estructura principal de programación en el sistema Arduino, ya que permiten utilizar y acceder a los recursos de las tarjetas electrónicas, ejecutar correctamente un programa, así como su comunicación con la computadora para desplegar información.



void setup() $\{...\}$

Esta función es del tipo **void**, no retorna ningún tipo de dato y representa la primera función a ejecutar en el sketch (sólo corre una vez); se utiliza para inicializar variables, configurar los recursos de la plataforma electrónica de los modelos de tarjetas como: puertos digitales entrada/salida, modo de operación y resolución de los convertidores analógico/digital, definir la velocidad de comunicación serie, etc.

void loop()
$$\{...\}$$

La función **loop()** es del tipo **void** y se procesa después de ejecutar la función **setup**. Incluye el código que estará corriendo de manera continua algoritmos de control, leyendo o escribiendo puertos digitales, activando salidas, adquiriendo datos de sensores a través de las entradas analógicas, etc.

La función **loop()** es el núcleo de todos los programas Arduino y realiza la ejecución del sketch de manera indefinida.



6.3.2 Utilidades

Para determinar cuántos bytes ocupa una variable de cualquier tipo, resulta de particular interés la siguiente función:

sizeof(dato)



Retorna el número de bytes del argumento **dato**, el cual puede ser del cualquier tipo (char, int, float, long, etc.) y para el caso de arreglos, entonces retorna el número de bytes ocupados en dicho arreglo.

6.3.3 Tipos de conversión

Dado una variable de cualquier tipo, ésta puede ser convertida a otro tipo de dato por medio de las siguientes funciones.

char(dato)

Convierte el argumento de entrada **dato** (que puede ser de cualquier tipo de dato) en una variable tipo char. Por ejemplo:

int j=9; //tipo de dato entero a convertir.

char c;//para registrar el tipo de dato que retorna.

c= char(j);//retorna un dato tipo char (el byte menos significativo de j).

byte(dato)

Convierte el argumento de entrada **dato** (que puede ser de cualquier tipo de dato) en una variable tipo byte.

int(dato)

Esta función convierte el argumento de entrada **dato** (que puede ser de cualquier tipo de dato) en una variable tipo entera (int).

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

Fernando Reyes Cortés • Jaime Cid Monjaraz

Alfaomega



Esta función convierte el argumento de entrada **dato** (que puede ser de cualquier tipo de dato) en una variable tipo entera (word).

La función **word(...)** también admite la siguiente sintaxis:



word(byte_high,byte_low)

Retorna una variable tipo **word** que se forma por medio de la concatenación de dos bytes, los argumentos de entrada representan el byte menos significativo **byte_low** (es el byte que se encuentra en el lado izquierdo de **word**) y **byte_high** es el byte más significativo de **word** (es el byte que se encuentra a la derecha).

long(dato)

Esta función convierte el argumento de entrada **dato** (que puede ser de cualquier tipo de dato) en una variable tipo entera larga (long).

float(dato)

Convierte el argumento de entrada **dato** (que puede ser de cualquier tipo de dato) en una variable tipo flotante (float).

A manera de ilustración, considere el siguiente ejemplo: sea el caso de tener una variable de tipo entero \mathbf{j} de 2 bytes y una variable de tipo flotante \mathbf{x} , cuando se desea convertir un entero a tipo flotante (4 bytes), se puede realizar de la siguiente forma:

int j=9; //tipo de dato entero a convertir.

float x;//para registrar el tipo de dato que retorna.

x = float(j); //convierte los 2 bytes de j en 4 bytes para x.

El valor que tiene la variable entera \mathbf{j} (dos bytes), cambia de formato a 4 bytes del tipo flotante, por medio de la función float(\mathbf{j}).

Alfaomega

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

Fernando Reyes Cortés • Jaime Cid Monjaraz



6.3.4 Funciones para puertos digitales entrada/salida

Las funciones que se encuentran relacionadas con la programación de los puertos digitales de las tarjetas Arduino (configurar modo de trabajo entradas/salidas), así como acceso a lectura/escritura de información son las que a continuación se describen.

pinMode(pin, mode)



Esta función configura de manera específica un **pin** (entero positivo) de un puerto digital, es decir, el número del conector del puerto digital; el argumento **mode** determina la forma de configuración del **pin**, ya sea como puerto de salida (**OUTPUT**), entrada (**INPUT**) o como entrada con resistencia interna de **pullup** (**INPUT_PULLUP**).

Por ejemplo, para configurar el pin 11 como puerto digital de salida se utiliza la siguientes sintaxis: pinMode(11, OUTPUT); este mismo puerto puede ser configurado como entrada de la siguiente manera: pinMode(11, INPUT).

Puertos digitales

Arduino configura de manera inicial todos los puertos digitales como entradas (INPUT), por lo que no se necesita una programación específica con la función **pinMode()**. De esta forma, el sistema queda protegido ante voltajes parásitos o ruido durante la fase transitoria del encendido o en alguna fase de reinicialización del sistema, ya que los puertos de entrada adquieren el estado de alta impedancia, esto se interpreta como si los puertos virtualmente no existieran, en otras palabras, como si dichos puertos no estuvieran conectados.

Resistencias de pullup

Es ampliamente recomendable colocar una resistencia de **pullup** a un pin de entrada, esto se realiza conectado un extremo de la resistencia (10 K Ω o 20 K Ω) y el otro extremo a + 5 Volts. Otra forma de hacerlo es conectar la resistencia a tierra **pull-down**. Varias tarjetas Arduino ya tienen integradas resistencias **pullup**, la forma de activarlas es la siguiente:

pin=12;//selecciona número de puerto. //Configura puerto digital como entrada. pinMode(pin, INPUT);//puerto entrada. //Activa resistencia pullup. digitalWrite(pin, HIGH); Es importante aclarar que la gran mayoría de las entradas analógicas de las tarjetas Arduino, pueden configurarse y utilizarse de la misma forma que los puertos digitales. Cada puerto digital cuenta con resistencia **pullup** de 20 K Ω integradas en el chip ATMEGA que pueden ser accedidas por software de la siguiente manera:



Las resistencias pullup se usan normalmente para conectar entradas como interruptores.



Cuando las terminales de los puertos digitales son configuradas como salidas (\mathbf{OUTPUT}), adquieren un estado de baja impedancia y pueden proporcionar hasta 40 mA a otros dispositivos o circuitos.

Cuando sucede un cortocircuito en las terminales de Arduino o se demanda corriente excesiva, se puede dañar o destruir la terminal de salida (o dañar el chip ATMEGA).

Cuando se conectan interfaces electrónicas a las tarjetas Arduino es ampliamente recomendable conectar a la terminal de salida (**OUTPUT**) resistencias de 470 Ω a 1 K Ω ; lo anterior protegerá los puertos de la tarjeta.

El puerto digital 13 tiene acoplado un LED integrado en el circuito impreso de las tarjetas Arduino y una resistencia limitadora de corriente. Cuando se activa por programación la resistencia interna pullup 20k para este pin, por ejemplo, **pinMode(13, INPUT)**; **digitalWrite(13, HIGH)**; el voltaje que se obtendrá será alrededor de 1.7 V en lugar de los 5 V, debido al consumo del LED y su resistencia bajan el nivel del voltaje, lo que se interpreta como un estado digital LOW. Por lo tanto, se recomienda utilizar una resistencia pull-down externa cuando este puerto está configurado como entrada.

digitalRead(pin)

Esta función retorna el estado digital (**HIGH** o **LOW**) de un **pin** específico de la terminal de puertos digitales, los cuales se encuentran comprendidos entre 0 al 13, estos valores se pueden especificar a través de variables o constantes. Por ejemplo, la siguiente sentencia lee el valor digital del puerto 12:

valor = digitalRead(12);

digitalWrite(pin, valor)

Escribe un nivel lógico (**HIGH** o **LOW**) asignado en la variable **valor** al número de puerto especificado por la variable **pin** cuyo rango puede ser del 0 al 13. La siguiente sentencia muestra cómo

Alfaomega

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

FERNANDO REYES CORTÉS • JAIME CID MONJARAZ

escribir un estado lógico HIGH en el puerto digital 10:

digitalWrite(10, HIGH); //salida estado lógico HIGH o 5 V en el pin 10.

♣ Ejemplo 6.1

Diseñar un sketch para enviar números consecutivos del 0 al 6 por medio de puertos de salida y recuperarlos a través de puertos de entrada. Exhibir el resultado en el monitor serial del ambiente de programación Arduino.

Solución

Enviar y recibir datos entre sistemas digitales es muy útil su empleo en robótica y mecatrónica; por ejemplo, para enviar a la consola de control del robot el número de sensores activos que se encuentran en el sistema mecánico, se requiere de un protocolo de comunicación el cual envíe datos por los puertos digitales de salida y sean recibidos en los puertos de entrada para su desplegado. Evidentemente esto requiere un preprocesamiento de manipulación de los bits de información para que se puedan desplegar correctamente.

El cuadro de código Arduino 6.1 muestra el algoritmo en lenguaje C del sketch **cap6_digitalports**, para programar 3 puertos digitales de entrada y 3 puertos como salida.

En la línea 1 se define la nomenclatura de los puertos a utilizar siendo los pins 5, 6, 7, 11, 12 y 13. El algoritmo utiliza la instrucción **for**(;;) {...} para asignar de manera periódica valores digitales a los puertos de salida y procesar la lectura en los puertos de entrada, para esta finalidad se emplean las variables pivote **i** y registro de resultado en **j** (ver línea 2).

La subrutina de configuración **setup()** se define en la línea 3 para establecer la velocidad de transmisión en 9600 Baudios y la configuración en modo de puertos de entrada digital para los pins 5, 6 y 7; puertos de salida los pins 11, 12 y 13. A partir de la línea 12 empieza el lazo principal de programación del sketch. La figura 6.1 muestra las conexiones eléctricas de los puertos entrada/salida en la tarjeta Arduino modelo UNO.

La línea 13 contiene la instrucción for (; ;) {...} con el pivote o índice i para enviar valores de 0 al 6 en los puertos digitales. Observe que se envían los primeros 3 bits menos significativos del pivote i, para esto se utiliza manipulación de bits a través del operador & (AND-lógica).

El bit menos significativo se transmite al puerto de salida pin13, por medio de la operación 0x0001&i; la variable entera i es de 2 bytes, es decir, el pivote i puede ser visto en memoria en forma binaria como la representación de 16 bits:

i=b15 b14 b13 b12 b11 b10 b9 b8 b7 b6 b5 b4 b3 b2 b1 b0

siendo b0 y b15 los bits menos y más significativos, respectivamente.

Por otro lado, la representación en memoria de la constante hexadecimal 0x0001 puede ser vista en formato binario como: 0x0001=0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1, por lo tanto la operación bit a bit **AND**-lógica proporciona el siguiente resultado:

$0x0001&i=0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0$

En otras palabras, se limpian con ceros los bits comprendidos entre el segundo bit menos significativo b1 hasta el más significativo b15, manteniendo exclusivamente el valor del bit menos significativo b0, el cual se envía al puerto digital pin13.

El segundo bit menos significativo se transmite en el pin12 por medio de la operación 0x0002&i, de manera análoga para el tercer bit menos significativo en el pin11 con 0x0004&i. Este procedimiento se lleva a cabo de las líneas 14 a la 16.

El procesamiento del resultado consiste en la manipulación de los bits de las lecturas en los puertos de entrada, de tal manera que se deben quedar acomodados los bits en forma adecuada, esto se realiza a partir de la línea 17 por medio de la lectura del bit menos significativo pin5, el cual mantiene el valor del bit menos significativo por medio de la operación 0x00001& digitalRead(pin5), registrando este resultado en la variable j.

El segundo bit menos significativo se obtiene en el pin 6, observe en este caso que primero se realiza un corrimiento a la izquierda << de este bit y posteriormente una operación bit a bit del tipo **OR**-lógica con los 16 bits de la variable **j**.

El último bit se obtiene del pin7, al cual se desplaza dos posiciones a la izquierda el bits menos significativo de esta lectura, para ubicarse en la tercera posición, después una operación **OR** usando el operador | para obtener el resultado final como se muestra en la línea 20:

j=0000 0000 0000 0 pin7 pin6 pin5

de esta forma, los 16 bits de la variable entera \mathbf{j} ya se encuentran en el formato adecuado para desplegar la información.

A manera de ejemplo en la tabla 6.5 se presenta un resumen de operaciones y resultados que se llevan a cabo en el cap6_digitalports.

Tabla 6.5 Manipulación de bits.

| Descripción | 0x0001 & i línea 14 del sketch cap6_digitalports . |
|-------------|---|
| Operación | 0000 0000 0000 0001 & b15 b14 b13 b12 b11 b10 b9 b8 b7 b6 b5 b4 b3 b2 b1 b0 |
| Resultado | 0000 0000 0000 000 ьо |
| Descripción | 0x0002 & i línea 15 del sketch cap6_digitalports . |
| Operación | 0000 0000 0000 0010 & b15 b14 b13 b12 b11 b10 b9 b8 b7 b6 b5 b4 b3 b2 b1 b0 |
| Resultado | 0000 0000 0000 00 b1 0 |
| Descripción | 0x0004 & i línea 16 del sketch cap6_digitalports . |
| Operación | 0000 0000 0000 0100 & b15 b14 b13 b12 b11 b10 b9 b8 b7 b6 b5 b4 b3 b2 b1 b0 |
| Resultado | 0000 0000 0000 0 b2 00 |
| Operación | j=0x0001&digitalRead(pin5); línea 17 del sketch cap6_digitalports . |
| Resultado | j=0000 0000 0000 000 pin5 |
| Operación | j=j digitalRead(pin6)<<1; línea 18 del sketch cap6_digitalports . |
| Resultado | j=0000 0000 0000 00 pin6 pin5 |
| Operación | $j=j \mid digitalRead(pin7) << 2; línea 20 del sketch \mathbf{cap6_digitalports}.$ |
| Resultado | j=0000 0000 0000 0 pin7 pin6 pin5 |



Manipulación de bits

Ejemplos adicionales sobre el uso de operadores lógicos y de corrimiento (izquierda y derecha) para manipulación de bits para diferentes variables de los tipos de datos del lenguaje C se presentan en el sitio Web de este libro.

```
○ Código Arduino 6.1: sketch cap6_digitalports
  Arduino. Aplicaciones en Robótica y Mecatrónica.
                                                        Disponible en Web
  Capítulo 6 Librerías y funciones Arduino.
  Fernando Reyes Cortés y Jaime Cid Monjaraz.
  Alfaomega Grupo Editor: "Te acerca al conocimiento".
   Sketch cap6_digitalports.ino
 1 int pin5=5, pin6=6, pin7=7, pin11=11, pin12=12, pin13=13;//asignación de puertos.
 2 int i, j;//variable i es el pivote de la instrucción for(; ;){...}, j registra resultado.
3 void setup() {//subrutina de configuración.
       Serial.begin(9600); //velocidad de transmisión 9600 Baudios.
       pinMode(pin5, INPUT); //pin 5 configurado como entrada digital.
       pinMode(pin6, INPUT); //pin 6 configurado como entrada digital.
 6
       pinMode(pin7, INPUT); //pin 7 configurado como entrada digital.
 7
       pinMode(pin11, OUTPUT); //pin 11 configurado como salida digital.
 8
       pinMode(pin12, OUTPUT); //pin 12 configurado como salida digital.
9
10
       pinMode(pin13, OUTPUT); //pin 13 configurado como salida digital.
11 }
12 void loop() {//lazo principal del sketch.
       for(i=0; i<7; i++)
13
         digitalWrite(pin13, 0x0001&i);//pin 13 con pin 5.
14
         digitalWrite(pin12, 0x0002&i);//pin 12 con pin 6.
15
         digitalWrite(pin11, 0x0004&i);//pin 11 con pin 7.
16
17
         j=0x0001&digitalRead(pin5);//lectura del pin 5 con máscara del primer bit.
         j=j|digitalRead(pin6)<<1;//lectura del pin 6 y corrimiento a la izquierda.
18
         //Los 16 bits de j tienen la forma: j=0000 0000 0000 0 pin7 pin6 pin5.
19
         j=j| digitalRead(pin7)<<2;//lectura del pin 7 y corrimiento a la izquierda.
20
         Serial.println("Salida,
                                   Entrada"); Serial.print(i); delay(10);
21
         Serial.print("\t \t "); delay(10);
22
23
         Serial.println(j); delay(10);
       \}// fin de la instrucción for (;;) \{...\}.
24
       Serial.println("En 5 segundos, se ejecutará una vez más el sketch");
25
       delay(5000);
26
27 }
```

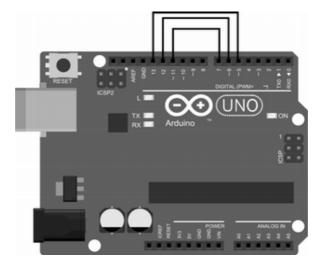


Figura 6.1 Conexión eléctrica de puertos digitales

La ejecución del sketch **cap6_digitalports** en la tarjeta Arduino produce el siguiente despliegue de resultados en la ventana del monitor serial del ambiente de programación Arduino (es muy importante abrir al máximo la ventana del monitor y activar la opción **desplazamiento** automático.

| Salida, | Entrada |
|---------|---------|
| 0 | 0 |
| Salida, | Entrada |
| 1 | 1 |
| Salida, | Entrada |
| 2 | 2 |
| Salida, | Entrada |
| 3 | 3 |
| Salida, | Entrada |
| 4 | 4 |
| Salida, | Entrada |
| 5 | 5 |
| Salida, | Entrada |
| 6 | 6 |
| | |

En 5 segundos, se ejecutará una vez más el sketch.

El letrero **Salida** indica la información enviada por los puertos configurados en modo salida (en este caso el pivote \mathbf{i}), mientras que, el letrero **Entrada** es la información procesada por el algoritmo del sketch que la almacena en la variable \mathbf{j} .

Los pasos requeridos para ejecutar el sketch **cap6_digitalports** en las tarjetas Arduino son los que se describen en la sección 2.4 del capítulo 2 **Instalación y puesta a punto del sistema Arduino**.

Por comodidad del lector, a continuación se presentan en forma sencilla y compacta dicho procedimiento:

- Descargar del sitio Web de este libro el código fuente en lenguaje C del sketch cap6_digitalports.
- En el menú **Herramientas** del ambiente de programación Arduino seleccionar el modelo de tarjeta electrónica Arduino.
- Establecer la velocidad de comunicación serial en 9600 Baudios.
- Compilar el sketch mediante el icono <equation-block>
- Descargar el código de máquina a la tarjeta Arduino usando 👈.
- Desplegar resultados con el monitor serial (activar con ${\cal P}$).
- Importante: para que los resultados del sketch cap5_apuntador se exhiban adecuadamente en el monitor serial del ambiente de programación, maximice la ventana y active la opción desplazamiento automático.



6.3.5 Funciones para entradas analógicas

El sistema Arduino contiene un grupo de funciones para acceder a los convertidores analógicos/digital (ADC's); dependiendo del modelo de la tarjeta Arduino existen un número determinado de canales ADC, por ejemplo en el modelo UNO existen 6 ADC's, 8 canales analógicos para la Mini y Nano, y 16 ADC's en el modelo Mega. En estos casos, la resolución de los ADC's es de 10 bits, por lo que las lecturas de voltaje se encuentran en el rango de $\frac{5}{1024}$ V, siendo la mínima lectura de 4.9 mV.

A continuación se describe la sintaxis de las funciones para entradas analógicas:



La función **analogRead(pin)** se emplea para realizar adquisición de datos de señales analógicas que entran al sistema en formato digitalizado con resolución de 10 bits (0 a 1023). La tabla 6.6 muestra el rango de valores del parámetro de entrada **pin**, el cual se encuentra en función del tipo de terminales analógicas que tenga asignado el modelo de tarjeta Arduino.

Tabla 6.6 Rango de valores del parámetro pin.

| Modelo de tarjeta Arduino | Rango de valores de pin |
|---------------------------|-------------------------|
| UNO | 0 a 5 |
| Mini | 0 a 7 |
| Nano | 0 a 7 |
| Mega | 0 a 15 |

Por la naturaleza de los DAC's, las terminales analógicas no requieren ser declaradas o inicializadas como INPUT como sucede con los puertos digitales entrada/salida.

Esta función retorna un número entre 0 y 1023. Cuando se realiza una lectura sobre una entrada analógica y ésta no se encuentra conectada a una carga eléctrica, el valor que retorna la función analogRead(...) puede fluctuar en función de un conjunto de características tales como ruido eléctrico, voltaje parásito, etc.



Esta función escribe una valor seudoanalógico usando la forma de modulación por ancho de pulso PWM (pulse width modulation) en la terminal de salida marcada como PWM sobre el circuito impreso de la tarjeta Arduino. En los modelos más recientes que emplean el chip ATMEGA328, el pin corresponde a: 3, 5, 6, 9, 10 y 11.

El valor puede ser especificado como una variable o constante de 8 bits comprendido entre 0 y 255. La señal PWM es una onda cuadrada que está directamente relacionada con el periodo de tiempo del pulso, un porcentaje de ese periodo, la señal estará en nivel bajo (LOW o 0V) y el resto del periodo estará en alto (HIGH o 5V).

La señal de onda tipo PWM se utiliza para variar la intensidad luminosa (brillo) de un LED, manejo de motores a pasos, control de velocidad en servos, disparo por cruce de cero en la etapa de potencia de hornos eléctricos, etc.

En la mayoría de los pins, la frecuencia de la señal PWM es aproximadamente de 490 Hz, pero para los pins 5 y 6 del modelo UNO y similares tiene una frecuencia de operación de 980 Hz, este mismo valor de frecuencia se conserva para el modelo Leonardo en los pins 3 y 11. Por lo tanto, el periodo de un ciclo de onda PWM es inversamente proporcional a la frecuencia de operación; para 490 Hz, el periodo corresponde a 2.04 mseg y para 980 Hz es 1.02 mseg.

- Las tarjetas Arduino que emplean los microcontroladores ATMEGA168 o ATMEGA328 los pins habilitados para trabajar con PWM son: 3, 5, 6, 9, 10, y 11; en las tarjetas están rotulados por símbolo \sim .
- En el modelo Mega corresponden a los pins 2 al 13 y del 44 al 46.
- Mientras que, el modelo Arduino Due soporta analogWrite(...) sobre los pins 2 al 13, más los pins DAC0 y DAC1, estos últimos pins son de naturaleza diferente a las señales PWM, ya que DAC0 y DAC1 son convertidores analógico digital y actúan como salidas reales analógicas.
- No se requiere llamar a la función **pinMode()** para configurar a estos pins como salida. La función **analogWrite(..)** no tienen nada que ver con los pins analógicos o con la función **analogRead(...)**
- Las salidas PWM generadas sobre los pins 5 y 6 tendrán ciclos de trabajo más altos que lo esperado, debido a que la interacción que tienen con las funciones millis(...) y delay(...) comparten el mismo timer interno usado para generar esas salidas PWM.



Señal PWM

PWM (Pulse Width Modulation) es un tipo de onda o señal cuadrada que se emplea en control digital, corresponde a un patrón de estados lógicos HIGH (5 Volts) y LOW (0 Volts), dado un periodo de la señal PWM se programa la porción del tiempo (ancho de pulso) que estará en HIGH y la parte proporcional en LOW. De esta forma, es posible modular el ancho de pulso que estará en los estados lógicos deseados.

La figura 6.2 muestra la forma común de una onda PWM usando el pin 3 y cuatro ciclos de trabajo considerando un periodo típico para las tarjetas Arduino aproximadamente de 2 mseg (corresponde a una frecuencia de 490 Hz), la función analogWrite(...) acepta valores entre 0 y 255.

Un valor de 0 representa el 100% de la señal en nivel LOW (evidentemente 0% en HIGH); mientras que, un valor de 64 significa 25% en estado HIGH y el 75% en nivel LOW, el valor 127 genera la mitad del periodo en 5 V y el 50% en 0 V, y 255 representa el 100% del ciclo de trabajo en HIGH, es decir 0% en estado bajo (LOW).

La función analogWrite(pin, valor) toma alrededor de 100 microsegundos para leer una entrada analógica, por lo que este dato establece la máxima razón de lectura, es decir, puede adquirir 10000 lecturas por segundo. El rango de entrada y la resolución puede ser cambiado usando la función analogReference(...).

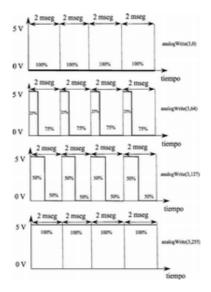


Figura 6.2 PWM: Modulación por ancho de pulso.

Cuando se ejecuta la función **analogWrite(pin, valor)** el **pin** correspondiente generará una onda cuadrada estacionaria con ciclo útil especificado por **valor** hasta que se modifique por otra ejecución de esta función o también por ejecutar a las funciones **digitalRead(...)** o **digitalWrite(...)** sobre el mismo **pin**.

La tabla 6.7 muestra la relación valor/salida de la señal PWM usando la función analogWrite(pin, valor), donde el argumento de entrada valor se encuentra en el rango de números enteros: $\in [0, 255]$.

Dependiendo del tipo de tarjeta, el periodo de la onda PWM corresponde a $2.04~\mathrm{mseg}$ ($490~\mathrm{Hz}$) y $1.02~\mathrm{mseg}$ ($980~\mathrm{Hz}$).

Tabla 6.7 Relación valor/salida PWM: analogWrite(pin,valor).

| valor | Relación de salida PWM |
|-------|--|
| 0 | Todo el periodo de la onda PWM en estado lógico \mathbf{LOW} (0 V). |
| 64 | $\frac{1}{4}$ del periodo en HIGH (5 V) y $\frac{3}{4}$ del periodo en LOW (0 V). |
| 127 | $\frac{1}{2}$ del periodo de la onda PWM en HIGH (5 V) y $\frac{1}{2}$ del periodo en LOW (0 V). |
| 192 | $\frac{3}{4}$ del periodo de la onda PWM en HIGH (5 V) y $\frac{1}{4}$ del periodo en LOW (0 V). |
| 255 | Todo el periodo de la onda PWM en HIGH (5 V). |



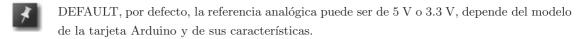
Esta función configura el voltaje de referencia usado por la entrada analógica, es decir, el valor usado que establece el límite superior de la entrada (generalmente es de 5 V o 3.3 V, dependiendo de las características de la tarjeta).

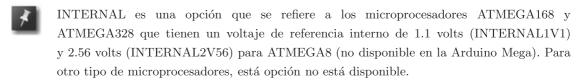
El argumento de entrada **tipo** establece la clase de referencia (nivel de voltaje que limita la magnitud de la entrada), la cual puede ser cualquiera de las siguientes opciones disponibles en el sistema Arduino: DEFAULT, INTERNAL, INTERNAL1V1, INTERNAL2V56, o EXTERNAL.

Alfaomega Arduino. Aplicaciones en Robótica y Mecatrónica

FERNANDO REYES CORTÉS • JAIME CID MONJARAZ

A continuación se explican las opciones disponibles:





EXTERNAL significa referencia externa, es un voltaje aplicado al pin AREF dentro del rango de 0 a 5 volts, el cual se emplea como una señal de voltaje de referencia (externo a la tarjeta).



- Cuando se cambia el valor de la referencia analógica, las primeras lecturas de la función analogRead(...) pueden ser inexactas.
- Como sugerencia, deseche las primeras diez lecturas.
- Cuando utilice voltaje de referencia externo en el pin AREF, no utilice referencias negativas (menores a 0 Volts) o mayores a 5 Volts, ya que pueden dañar la tarjeta electrónica.
- Si está usando referencia externa en el pin AREF, entonces debe utilizar la opción de referencia analógica EXTERNAL en analogReference(EXTERNAL), antes de utilizar la función analogRead(...).
- De no ser así, se podría producir un cortocircuito entre el voltaje de referencia interno y el que se encuentra en el pin AREF, cuya posible consecuencia es que se dañe la tarjeta electrónica.



Es ampliamente recomendable conectar el voltaje de referencia externo como divisor de voltaje, para esto se sugiere una resistencia de 32 K Ω entre el pin AREF y la terminal del voltaje externo. Esta resistencia alterará el voltaje usado como referencia, hay que tomar en cuenta que de manera interna en la tarjeta electrónica también se encuentra una resistencia del mismo valor de 32 K Ω asociada al pin AREF, las dos resistencias actúan como un divisor de voltaje.

Por ejemplo, suponga que el voltaje externo es de 5 V, entonces el divisor de voltaje sobre el pin AREF será $V_{\rm AREF}$:

$$V_{\rm AREF} = 5 \mathrm{V} \frac{32 \mathrm{~K}\Omega}{32 \mathrm{~K}\Omega + 32 \mathrm{K}\Omega} = 2.5 \mathrm{V}.$$

De esta forma, es posible conmutar de manera segura entre los voltajes de referencia externo e interno.



analogWriteResolution(num_bits)

Establece la resolución en bits de la función **analogWrite(...)** a través del argumento de entrada **num_bits**, cuyo rango de valores se encuentra entre 1 y 32. Cuando se selecciona una resolución mayor o menor a la capacidad del modelo de tarjeta, el valor usado en **analogWrite(...)** será truncado al valor más alto o puesto en cero, respectivamente. Por defecto es de 8 bits, por lo que **num_bits** adquiere valores comprendidos entre 0 y 255; sin embargo, la resolución depende de las características de la tarjeta Arduino; por ejemplo, para el modelo UNO es de 8 bits, por otra parte, el modelo Due puede ser programado a 12 bits, en este caso **num_bits** toma valores de 0 a 4095. La función **analogWriteResolution(...)** no retorna ningún tipo de resultado.



6.3.6 Características de los pins de entradas analógicas

A continuación se describe los pins de entradas analógicas de las tarjetas Arduino que utilizan los microcontroladores ATMEGA8, ATMEGA168, ATMEGA328 y ATMEGA1280.

Los convertidores analógico/digital de esos microcontroladores contienen seis canales con una resolución de 10 bits, es decir retornan enteros entre 0 y 1023. Estos seis canales también pueden trabajar como puertos digitales, es decir tienen las mismas funciones que los pins 0 al 13, esto tiene la ventaja de que cuando la aplicación demanda más puerto digitales y los canales analógicos no se encuentran en uso, entonces dichas entradas analógicas pueden ser configuradas como puerto

digitales.

Configuración de entradas analógicas como puertos digitales

Existen constantes predefinidas (A0, A1, A2, A3, A4 y A5) para referenciar a los seis canales analógicos; la tabla 6.8 muestra la nomenclatura o notación de dichas constantes, así como el número de pin de cada canal analógico.

Tabla 6.8 Constantes para canales analógicos.

| Notación | Número de pin | Descripción |
|----------|---------------|-------------------|
| A0 | 14 | canal analógico 0 |
| A1 | 15 | canal analógico 1 |
| A2 | 16 | canal analógico 2 |
| A3 | 17 | canal analógico 3 |
| A4 | 18 | canal analógico 4 |
| A5 | 19 | canal analógico 5 |

Las siguientes instrucciones permite configurar al canal analógico 0 como puerto digital de salida y ponerlo en estado lógico HIGH:

pinMode(A0, OUTPUT);//configura el canal analógico 0 como puerto de salida. digitalWrite(A0, HIGH);//puerto digital de salida en estado lógico HIGH.

Resistencias de pullup en canales analógicos

Los pins analógicos tienen resistencias de pullup, las cuales trabajan de la misma forma que las resistencias de los puertos digitales. Para habilitar las resistencias de pullup en los canales analógicos se utiliza el siguiente código:

digitalWrite(A0, HIGH);//habilita resistencia de pullup en el canal analógico 0.



Debe tomarse en cuenta que habilitar las resistencias de pullup afectará los valores de lectura de la función **analogRead()**.



La función analogRead(...) no trabajará correctamente, si su correspondiente pin ha sido previamente configurado como puerto digital de salida; si este es el caso, hay que habilitarlo como entrada (por ejemplo, el canal 0: pinMode(A0, INPUT);) antes de usar la función analogRead(...).



De manera similar, si el pin ha sido configurado como puerto de salida y fue habilitado en estado lógico HIGH, entonces la resistencia de pullup estará activada, por lo que deberá ser nuevamente configurado como entrada (por ejemplo, el canal 0: pinMode(A0, INPUT);).



Una vez que se ha restaurado la configuración en modo entrada, es recomendable esperar al menos 20 mseg antes de realizar lecturas, posteriormente desechar las primeras diez lecturas de analogRead(...).



Adquisición de señales analógicas

Varios ejemplos de adquisición de datos y señales analógicas se presentan en el sitio Web de este libro; se ilustra el empleo de las instrucciones AnalogRead(...), pinMode(...), analogWriteResolution(...), digitalWrite(...) y analogReference(...), para sensar señales analógicas y activar interruptores mecánicos y electrónicos.

También se presenta la forma de configurar la resolución del convertidor analógico digital de 10 o 12 bits y sus respectivas referencias de voltaje, estas características dependen de la clase de modelo Arduino.



Termómetro

Varios ejemplos sobre el desarrollo de termómetros o medidores de temperatura se analizan en el sitio Web de la presente obra. Se discute el código fuente en lenguaje C y la forma de calibrar diversos sensores de temperatura, así como el desplegado de la temperatura en °C, °F, °K, mediante exhibidores conectados a la tarjeta Arduino.

& Ejemplo 6.2

Rediseñar el ejemplo 6.1 programando las señales de entrada analógicas $A0, A1, \dots, A5$ como puertos digitales entrada/salida. Proponer otro esquema de procesamiento de la información de la entrada empleando unión de datos en lugar de corrimientos y operaciones lógicas de bits.

Solución

El cuadro de código Arduino 6.2 contiene la descripción y documentación técnica del sketch cap6_digitalportsUnio, el cual posee un algoritmo diferente para el procesamiento de información escritura/lectura de los puertos digitales presentado en el cuadro 6.1 sketch cap6_digitalports del ejemplo 6.1.

Las conexiones eléctricas de los pins analógicos se describen en la figura 6.3, el pin A0 se conecta con el pin A5, el pin A4 con el pin A1 y los pins A3 y A2 se encuentran unidos. Observe que a partir de la línea 6 del cuadro de código 6.2 se lleva a cabo la declaración de la unión de datos para las variables **datoSalida** y **datoEntrada**. Esta unión de datos está formada por dos campos, uno destinado a una variable de tipo entera k y el otro a una estructura de datos **datosbits** conformada por 8 campos con longitud de un bit cada uno. La ventaja que poseen las uniones de datos es que comparten la misma localidad de memoria todos sus campos. De tal forma que la modificación de un solo campo de bit en **datosbits**, automáticamente se refleja en el campo k.

En la línea 20 inicia la subrutina de configuración **setup()** donde se establece la velocidad de comunicación serial en 9600 Baudios y se configuran los pins A0, A1 y A2 como puertos de entrada digitales y A3, A4 y A5 como puertos de salida.

Note que no es necesario declarar a las constantes A0, A1, \cdots , A5; esto ya lo realiza de manera interna el sistema Arduino. La línea 6.33 marca el inicio del lazo principal de programación **loop()** del sketch.

La instrucción for (; ;){...} de la línea 30 inicializa en cero el campo k de la variable datoSalida, debido a las características de la unión de datos, las líneas 31-33 se escriben en los puertos de salida A0, A1, A2 los tres bits menos significativos correspondientes a los campos: datoSalida.datosbits.bit0, datoSalida.datosbits.bit1 y datoSalida.datosbits.bit2, respectivamente. Mientras que en las líneas 34 a la 35 se realiza la lectura de dichos bits.

En contraste con el ejemplo 6.1 donde se utilizan operadores lógicos y desplazamiento de bits, con este opcional procedimiento presenta mayores ventajas debido a que la unión de datos no requiere acomodar los bits, por lo que es mucho más sencillo y claro.

⊙ Código Arduino 6.2: sketch cap6_digitalportsUnio

Arduino. Aplicaciones en Robótica y Mecatrónica.

Disponible en Web



Capítulo 6 Librerías y funciones Arduino.

Fernando Reyes Cortés y Jaime Cid Monjaraz.

Alfaomega Grupo Editor: "Te acerca al conocimiento".

Sketch cap6_digitalportsUnio.ino

```
1 //Declaración de la variable tipo unión con sus campos de datos.
```

- 2 // Se tienen las variables datoSalida y datoEntrada del tipo unión.
- 3 //El campo destinado a la variable entera k comparte la misma localidad de memoria

```
4 //con el campo de la estructura datosbits, cualquier modificación a nivel de bit en
 5 //los campos bit0 al bit7, modifica el valor del campo entero k.
 6 union{
       int k;
 7
8
       struct{
          int bit0: 1;//bit menos significativo, campo de un bit.
 9
10
         int bit1: 1;//campo de un bit.
11
         int bit2: 1;//campo de un bit.
         int bit3: 1;//campo de un bit.
12
         int bit4: 1;//campo de un bit.
13
         int bit5: 1;//campo de un bit.
14
         int bit6: 1;//campo de un bit.
15
         int bit7: 1;//bit más significativo, campo de un bit.
16
17
       } datosbits;// campo de 8 bits.
18 } datoSalida, datoEntrada;
19 //Subrutina de configuración.
20 void setup(){
       Serial.begin(9600);//velocidad de transmisión 9600 Baudios.
21
       pinMode(A0, INPUT);//pin analógico A0 configurado como entrada digital.
       pinMode(A1, INPUT);//pin analógico A1 configurado como entrada digital.
23
       pinMode(A2, INPUT);//pin analógico A2 configurado como entrada digital.
24
       pinMode(A3, OUTPUT);//pin analógico A3 configurado como salida digital.
25
       pinMode(A4, OUTPUT);//pin analógico A4 configurado como salida digital.
26
       pinMode(A5, OUTPUT);//pin analógico A5 configurado como salida digital.
27
28 }
```

```
Continúa código Arduino 6.2a: sketch cap6_digitalportsUnio
  Arduino. Aplicaciones en Robótica y Mecatrónica.
                                                       Disponible en Web
  Capítulo 6 Librerías y funciones Arduino.
  Fernando Reyes Cortés y Jaime Cid Monjaraz.
  Alfaomega Grupo Editor: "Te acerca al conocimiento".
   Continuación del sketch cap6_digitalportsUnio.ino
29 void loop(){//lazo principal del sketch.
       for(datoSalida.k=0; datoSalida.k<7; datoSalida.k++){
30
          digitalWrite(A5, datoSalida.datosbits.bit0);//A5 con A0.
31
32
          digitalWrite(A4, datoSalida.datosbits.bit1);//A4 con A1.
          digitalWrite(A3, datoSalida.datosbits.bit2);//A3 con A2.
33
          datoEntrada.datosbits.bit0=digitalRead(A0);
34
          datoEntrada.datosbits.bit1=digitalRead(A1);
35
          datoEntrada.datosbits.bit2=digitalRead(A2);
36
          Serial.println("Salida,
                                    Entrada");
37
38
          Serial.print(datoSalida.k);
39
          delay(10);
          Serial.print("\t \t \);
40
          delay(10);
41
          Serial.println(datoEntrada.k);
42
          delay(10);
43
       }//fin de la instrucción for(;;){...}.
44
       Serial.println("En 5 segundos, se ejecutara una vez más el sketch");
45
       delay(5000);
46
47 }
48 //Ver instrucciones para ejecutar este sketch en la página 148.
```

Las líneas 38 y 42 envían el resultado (datos salida/entrada) al monitor serial para verificar que efectivamente se trata del mismo dato procesado. El resultado que presenta este sketch en el monitor serial del ambiente de programación Arduino es exactamente el mismo del ejemplo 6.1.

En la página 148 se encuentran los pasos requeridos para descargar el código de máquina y ejecutar el sketch **cap6_digitalportsUnio** en las tarjetas Arduino.



Figura 6.3 Conexión eléctrica de los pins analógicos A0, A1, · · · , A5.

♣ ♣ Ejemplo 6.3

Desarrollar un sketch para convertir una onda senoidal de amplitud unitaria y periodo de 6 segundos en onda cuadrada tipo PWM.

Solución

Dentro de las aplicaciones que tiene la señal PWM se encuentra control de potencia para motores de CA, control de motores a pasos, detectores por cruce de cero, control On/Off, conversión de señales senoidales analógicas a ondas cuadradas

Considere una señal senoidal de amplitud unitaria y de periodo 6 segundos, la cual se desea convertir en una onda tipo PWM; varias aplicaciones de control automático requieren convertir señales periódicas en ondas cuadradas conservando el mismo periodo de la señal.

Por las características que tiene la onda senoidal, el algoritmo de conversión consiste en detectar el signo de la amplitud, es decir, cuando sea positiva, se genera el pulso en alto (HIGH) y para amplitudes cero y negativas, se genera un pulso en bajo (low), de esta forma la correspondiente onda PWM conserva el mismo periodo de la onda senoidal.

La línea 1 declara el número de pin (puerto 3) donde se generará la señal PWM, en la línea 2 la

variable **dato** de tipo flotante se utiliza para registrar la información de la señal senoidal y para analizar su signo; la variable **t** contabiliza el tiempo por incrementos de 2 mseg (línea 3).

La subrutina o función de configuración **setup()** inicia en la línea 4, la cual establece la velocidad de comunicación serial en 9600 Baudios para envío/recepción de información entre la computadora y la tarjeta Arduino, la configuración del pin 3 como puerto digital de salida. El lazo principal de programación **loop()** empieza a partir de la línea 8.

La función senoidal se programa en la línea 9, la cual tiene amplitud unitaria y frecuencia de $\frac{1}{6}$ Hz o 6 segundos de periodo. El signo de esta función se determina en la línea 10; si es positivo, entonces se mantiene en alto (HIGH) el estado del puerto pin 3, y si el signo de la amplitud es negativo corresponde a cero (LOW) como se muestra en la línea 13.

Observe que la activación de la señal PWM se realiza mediante la función **analogWrite(...)**; el incremento del tiempo se realiza cada 2 milisegundos (línea 15).

La figura 6.4 muestra la forma de convertir la señal senoidal sen $(2\pi \frac{t}{6})$ a señal tipo PWM; el lector puede modificar la frecuencia de la onda senoidal y comprobar en un osciloscopio la forma de la onda PWM.

Para ejecutar el sketch **cap6_pwm** en la tarjeta Arduino puede consultar el procedimiento descrito en la página 148.

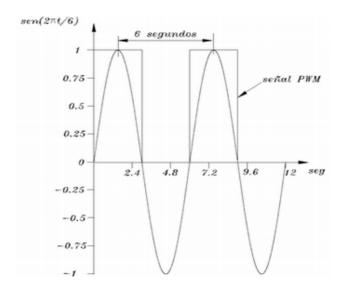


Figura 6.4 Conversión de onda senoidal sen $(2\pi \frac{t}{6})$ a PWM.

Código Arduino 6.3: sketch cap6_pwm Arduino. Aplicaciones en Robótica y Mecatrónica. Disponible en Web Capítulo 6 Librerías y funciones Arduino. Fernando Reyes Cortés y Jaime Cid Monjaraz. Alfaomega Grupo Editor: "Te acerca al conocimiento". Sketch cap6_pwm.ino 1 int pin3 = 3;//pin 3 señal PWM. 2 float dato;//registra la información de la función senoidal. 3 float t=0;//contabiliza el tiempo. 4 void setup(){//subrutina de configuración. Serial.begin(9600);//Velocidad de transmisión en 9600 Baudios. pinMode(pin3, OUTPUT);//puerto de salida. 6 7 } 8 void loop(){//lazo principal de programación del sketch. dato= $\sin(2*M_PI*t/6.0)$;//función senoidal $\sin(2\pi \frac{t}{\epsilon})$. 10 if (dato >=0)//analiza signo de la función senoidal. 11 analogWrite(pin3, 255);//signo positivo genera señal PWM en HIGH. else//signo negativo. 12 analogWrite(pin3, 0);//signo negativo, genera señal PWM en LOW. 13 14 delay(2);t=t+0.002;//incremento del tiempo: t=t+0.002. 15 16 }//Para ejecutar este sketch, ver la página 148.

♣ ♣ ♣ Ejemplo 6.4

Desarrollar un sketch para adquisición de señales analógicas.

Solución

La adquisición de datos es un proceso básico para las tarjetas Arduino; la figura 6.5 muestra un divisor de voltaje formado por dos resistencias iguales de 10 K Ω , denotadas por R_1 y R_2 y un potenciómetro P de 1 K Ω , también se cuenta con una fuente de alimentación de 1.5 V (batería o pila tipo AA). El cursor del potenciómetro se encuentra conectado al canal analógico 0 (A0) de la tarjeta Arduino UNO, dando una lectura de $\frac{P+R_2}{R_1+R_2+P}$ 1.5 V, es decir el voltaje de adquisición puede variar entre 0.75 V (cuando P=0 Ω) a 0.825 V (si P= 1 K Ω).

Alfaomega Arduino. Aplicaciones en Robótica y Mecatrónica

Fernando Reyes Cortés • Jaime Cid Monjaraz

El cuadro de código Arduino 6.4 presenta el sketch **cap6_adqDatos** con la programación en lenguaje C requerida para realizar la adquisición de datos de lectura para la señal analógica del potenciómetro o de un sensor analógico; este programa ilustra la forma de capturar señales y puede ser adaptado a diversos problemas de automatización.

En la línea 1 se declara la variable de tipo entero **dato_lectura** para registrar las lecturas de la función **analogRead(...)**. Recuerde que esta función retorna datos de tipo entero (int) comprendidos en el intervalo de 0 a 1023, los cuales corresponden a 10 bits ($2^{10} = 1024$, como se incluye el valor de 0 V, entonces el rango va de 0 a 1023).

En la línea 2 se declara la variable de tipo flotante **voltaje** para almacenar la conversión de la lectura entera a voltaje, es decir registrar la medición de la señal ya calibrada (ver línea 7, adquisición de datos).

El factor de conversión está en función de la resolución del convertidor analógico, depende del voltaje de referencia (en este caso, el valor por defecto es 5 V) y el número de bits que está utilizando el convertidor analógico/digital es 10 bits, por lo que se tienen $2^{10}=1023$ niveles de cuantificación en el proceso de la conversión, por lo que la resolución de la medición de la señal está determinada por: $\frac{5}{1023}=0.0048875$ V o 4.8875 mV; la resolución es la mínima medición que puede discernir o detectar un instrumento de medición y se puede mejorar aumentando el número de bits o disminuyendo el voltaje de referencia.

Cuando el voltaje de referencia se disminuye, es necesario acoplar (atenuar) la señal a procesar por medio de amplificadores de instrumentación, de tal forma que pueda entrar en el rango de voltaje de la señal de referencia.

Observe que en la función de configuración **setup()** (línea 3) no se requiere programar los canales analógicos como puertos de entrada, ya que se encuentran predefinidos de esa forma. Sólo se establece la comunicación serial en 9600 Baudios. El lazo principal de programación **loop()** de sketch inicia en la línea 6.

En la línea 7 se realiza la adquisición de datos del voltaje del potenciómetro usando la función analogRead(A0) (la constante A0 ya está predefinida, y por lo tanto, no es necesario declararla) se registra la lectura en la variable dato_lectura, la conversión a voltaje se lleva a cabo en la línea 10 desplegando su resultado con cuatro fracciones en el monitor serial del ambiente de programación Arduino (línea 12). Después de 100 mseg se ejecuta una vez más el sketch.

La ejecución del sketch **cap6_adqDatos** en las tarjetas Arduino es similar al procedimiento descrito en la página 148.

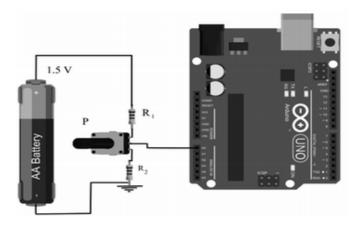


Figura 6.5 Adquisición de datos, canal analógico A0 del modelo Arduino UNO.

○ Código Arduino 6.4: sketch cap6_adqDatos Arduino. Aplicaciones en Robótica y Mecatrónica. Disponible en Web Capítulo 6 Librerías y funciones Arduino. Fernando Reyes Cortés y Jaime Cid Monjaraz. Alfaomega Grupo Editor: "Te acerca al conocimiento". Sketch cap6_adqDatos.ino 1 int dato_lectura = 0;// variable para almacenar el valor de lectura. 2 float voltaje;//variable para convertir la lectura a voltaje. 3 void setup(){ Serial.begin(9600);//velocidad de transmisión en 9600 Baudios. 5 } 6 void loop(){//adquisición de datos. dato-lectura = analogRead(A0);//Lectura del canal analógico A0. 7 Serial.print("Lectura="); 8 Serial.println(dato_lectura); //lectura (número entero entre 0 y 1023). 9 $voltaje = 0.0048875* dato_lectura; // convierte lectura a voltaje. \\$ 10 Serial.print("Voltaje="); 11 Serial.println(voltaje,4); //despliega valor del voltaje con 4 fracciones. 12 13 delay(100);// cien milisegundos de espera. 14 }//Para ejecutar este sketch, consulte la página 148.



6.3.7 Funciones time

El sistema Arduino contiene cuatro funciones que proporcionan información o procesan el tiempo como variable de medición (a través de timers).

millis()



Retorna el número de milisegundos que han transcurrido desde que el sketch o el programa Arduino empezó a ejecutarse en la tarjeta electrónica. El tipo de dato que retorna es del tipo **long** sin signo (unsigned long) y no lleva argumento de entrada. El resultado que retorna se va a cero (*overflow*) después de 50 días. Por ejemplo,

unsigned long valor; //tipo de dato que retorna.

valor = millis();//retorna el número de milisegundos que han transcurrido.

Nota: un segundo contiene mil milisegundos, es decir: 1 segundo= 1000 milisegundos.

micros()



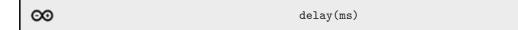
Retorna el número de microsegundos que han transcurrido desde que se ejecutó el programa o sketch en la tarjeta Arduino, el tipo de dato que retorna es unsigned long y no contiene argumento de entrada. Este valor puede ir a cero (overflow) después de 70 minutos. Para los modelos de 16 Mhz como Duemilanove y Nano tiene una resolución de 4 microsegundos, por lo que el valor que retorna siempre es múltiplo de 4. Para las tarjetas de 8 Mhz como LilyPad, la resolución de esta función es de 8 microsegundos.

Por ejemplo,

unsigned long valor; //tipo de dato que retorna.

valor = micros();//retorna el número de microsegundos que han transcurrido.

- Un milisegundo contiene 1000 microsegundos, 1 mseg = 1000 μ seg.
- Un millón de microsegundos forman a un segundo, 1 μ seg = 10^{-6} segundos.
- 1 segundo =1000 mseg = 1000000 μ seg.



Genera retardo de tiempo o pausa por un número determinado de milisegundos especificado en el argumento de entrada ms de tipo unsigned long. Esta función no retorna datos. Por ejemplo,

delay(1000); //genera una pausa de 1000 mseg o un segundo.



delayMicroseconds(us)

Genera una pausa en el programa por la cantidad de tiempo expresado en microsegundos en el argumento de entrada **us**, el cual es del tipo entero sin signo (**unsigned int**). Esta función no retorna datos.

Observe que debido a que el argumento de entrada **us** es del tipo unsigned int, entonces el valor más grande del argumento de entrada que produce un retardo exacto es 16383. Para generar retardos de tiempo mayores se debe usar la función **delay(...)**. Por ejemplo:

unsigned int us=1000; //tipo de argumento de entrada. delayMicroseconds(us);//pausa por 1000 microsegundos o un milisegundo.



6.3.8 Funciones matemáticas

Arduino tiene algunas funciones matemáticas que no requieren utilizar en la cabecera o *header* del sketch la directiva #include <math.h>, entre dichas funciones se encuentran las siguientes:



Calcula el mínimo de dos números **x**,**y** de cualquier tipo de datos y devuelve el número más pequeño. El tipo de dato que retorna corresponde a la misma naturaleza del argumento mínimo.

Ejemplo:

float x=100.3, y=201.34;

float valor;//retorna el mismo tipo de dato del argumento mínimo.

valor=min(x, y); //retorna 100.3.

Alfaomega

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

Fernando Reyes Cortés • Jaime Cid Monjaraz

Por la forma en que ha sido implementada la función **min(...)** evite usar como argumentos: operadores u otras funciones, ya que produce resultados incorrectos.

Por ejemplo:

valor=min(x*x, sin(x));//retorna un resultado incorrecto.

Es preferible el siguiente código:

 $y = \sin(x);$

x=x*x;

valor=min(x, y);//se recomienda usar este código.

max(x,y)



Calcula el máximo de dos números \mathbf{x} , \mathbf{y} de cualquier tipo de datos y devuelve el número más grande. El resultado que devuelve es del mismo tipo del argumento máximo.

Ejemplo:

float x=100.3, y=201.34;

float valor;//retorna el mismo tipo de dato del argumento máximo.

valor=max(x, y); //retorna 201.34.

El código en lenguaje C de la función max(...) retorna resultados incorrectos cuando los argumentos tienen operadores u otras funciones.

Por ejemplo:

valor=max(x++, y--);//retorna un resultado incorrecto.

Es preferible el siguiente código:

y++;

x=x--;

valor=max(x, y);//se recomienda usar este código.

abs(x)



Obtiene el valor absoluto de un número \mathbf{x} . Retorna \mathbf{x} si es mayor o igual que cero, $-\mathbf{x}$ si es menor que cero. El argumento \mathbf{x} puede ser de cualquier tipo de dato válido del lenguaje \mathbf{C} .

Por la forma en que ha sido implementada la función **abs(...)** evite usar como argumentos operadores u otras funciones, ya que produce resultados incorrectos.

Por ejemplo:

valor = abs(x*x*x); //retorna un resultado incorrecto.

Es preferible el siguiente código:

x=x*x*x:

valor=abs(x);//se recomienda usar este código.



Restringe un número \mathbf{x} dentro de un rango determinado por el límite inferior \mathbf{a} y límite superior \mathbf{b} , es decir: $x \in [a, b]$. Los argumentos \mathbf{x} , \mathbf{a} y \mathbf{b} pueden ser de cualquier tipo de datos.

 $\text{Retorna:} \left\{ \begin{array}{l} \mathbf{x} \text{ si } \mathbf{x} \text{ se encuentra entre } \mathbf{a} \text{ y } \mathbf{b}, \text{ es decir si } x \in [a,\ b]. \\ \mathbf{a} \text{ si } \mathbf{x} \text{ es menor que } \mathbf{a},\ x < a. \\ \mathbf{b} \text{ si } \mathbf{x} \text{ es mayor que } \mathbf{b},\ x > b. \end{array} \right.$



Transforma un número \mathbf{x} desde un rango $[\mathbf{a}, \mathbf{b}]$ hacia otro rango $[\mathbf{c}, \mathbf{d}]$, es decir, si \mathbf{x} se encuentra en el intervalo definido por $x \in [a, b]$, entonces esta función lo transforma o mapea hacia otro intervalo definido en $x \in [c, d]$, en otras palabras: $x \in [a, b] \to [c, d]$.

Los argumentos de entrada son del tipo de datos enteros largos (long):

- ${f x}$ es el número a mapear y puede ser de cualquier tipo de dato.
- ${\bf a}$ límite inferior del intervalo actual de ${\bf x}.$
- ${\bf b}$ límite superior del intervalo actual de ${\bf x}.$
- ${f c}$ límite inferior del nuevo intervalo para ${f x}.$
- \mathbf{c} límite superior del nuevo intervalo para \mathbf{x} .

Retorna un tipo de dato entero largo (long) correspondiente al valor transformado en el intervalo deseado, por ejemplo:

$$y=map(x, 0, 255, 0, 1023);$$

La función map(...) no restringe los límites de los intervalos, por lo que se puede tener que los

Alfaomega

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

Fernando Reyes Cortés • Jaime Cid Monjaraz

límites inferiores sean mayores que los límites superiores. Por ejemplo:

```
y=map(x, 0, 50, 0, 50);//intervalos en reversa.
```

También se puede tener el caso de límites negativos, como en el siguiente ejemplo:

```
y=map(x, 0, 50, 50, -50);//intervalos en reversa.
```

Debido a que la función map(...) trabaja con números enteros largos (long), entonces no puede retornar datos de tipo flotante.

El código ejemplo 6.2 muestra cómo está implementada la función map(...).

Código ejemplo 6.2

```
Código de la función map(...)
```

```
long map(long x, long a, long b, long c, long d){
 return (x-a)*(d-c)/(b-a)+c;
```



Aplicaciones de la función map(...)

La función map(...) es interesante para aplicaciones de ingeniería y ciencias exactas, ya que periódicamente se requiere convertir, calibrar y limitar señales de instrumentación de un espacio hacia otro. Tal es el caso de algunos algoritmos de control que no tienen estructura matemática que sature la energía enviada al servoamplificador; este es caso del PD, entonces se requiere de una transformación hacia un espacio saturado con valores de retorno flotante y double. En el sitio Web, de presente obra se ilustran aplicaciones de la función map(...).



pow(base, exponente)

La función **pow(base, exponente)** calcula el número **base** elevado a la potencia **exponente**, es decir realiza la siguiente operación base exponente.

Los argumentos **base** y **exponente** son del tipo de datos flotante y por lo tanto, **exponente** puede representar una potencia fraccionaria.

La función **pow(...)** retorna un número del tipo double. Por ejemplo:

double y, y1;

y=pow(2, 10);// significa 2^{10} .

y1=pow(3.1416, 2.345);// significa $\pi^{2.345}$.



sqrt(x)

Calcula la raíz cuadrada de un número positivo o cero \mathbf{x} , el argumento de entrada \mathbf{x} es un dato de cualquier tipo. Retorna la raíz cuadrada (no negativa, $+\sqrt{x}$) como un dato de tipo double. Por ejemplo:

double y;

y=sqrt(3.33);// significa $\sqrt{3.33}$.

Observe que la raíz cuadrada también se puede calcular de la siguiente forma:

double y;

y=pow(3.33, 0.5);// significa $\sqrt{3.33} = (3.33)^{0.5} = (3.33)^{\frac{1}{2}}$.



sin(theta)

Función trigonométrica senoidal, calcula el seno de un ángulo θ , el cual se encuentra expresado en radianes.

El argumento θ (theta) es del tipo flotante y el resultado es un número del tipo double, el cual se encuentra acotado dentro del intervalo -1 a 1. Por ejemplo,

double y;

float theta=3.1416;

y=sin(theta); //obtienen la función $sin(\theta)$.

Alfaomega

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

Fernando Reyes Cortés • Jaime Cid Monjaraz

cos(theta)



Calcula el coseno de un ángulo θ expresado en radianes. El argumento θ (**theta**) es del tipo flotante y el resultado es un número del tipo double, el cual se encuentra acotado dentro del intervalo -1 a 1. Por ejemplo,

```
double y;
float theta=0.234;
y=cos(theta); //obtienen la función cos(\theta).
```

tan(theta)



Calcula la tangente de un ángulo θ expresado en radianes. El argumento θ (**theta**) es del tipo flotante y el resultado es un número del tipo double, el cual se encuentra en el intervalo de $-\infty$ a ∞ . Por ejemplo, el ejemplo del cuadro 6.3 describe el uso de la función tangente:

€ Código ejemplo 6.3

Cálculo de la función tangente.

```
void setup() {
        Serial.begin(9600);//abre puerto serial en 9600 Baudios.
}
void loop(){
    float x, theta=M_PI;
    x=tan(theta);//calcula la tangente de theta.
        Serial.println(x);//imprime resultado.
        delay(10);//pausa por 10 milisegundos.
}
```

random(min, max)



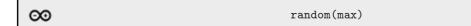
La función random(min, max) genera números seudoaleatorios, donde el argumento min es un número de tipo long y representa la cota más pequeña del valor aleatorio, este argumento es opcional y el argumento max (long) es obligatorio, denotando la cota máxima del valor aleatorio.

Retorna un número aleatorio de tipo long entre min y max-1.

Por ejemplo, para generar números aleatorios entre 5 y 9, se puede usar la función **random(....)** de la siguiente forma:

numero_rand=random(5, 10);//números aleatorios entre 5 y 9.

Debido a que el argumento **min** es opcional, la función **random(...)** puede tener la siguiente sintaxis:



Cuando no se indica el argumento **min**, entonces éste por default adquiere el valor de cero y el argumento **max** especifica la cota máxima de los números aleatorios, por lo que el resultado se encuentra en el intervalo de 0 a **max-1**. Por ejemplo:

numero_rand=random(10);//números aleatorios de 0 a 9.



La función **randomSeed(...)** inicializa el seudogenerador de números aleatorios, empezando en un punto arbitrario de una secuencia aleatoria. El argumento de entrada es del tipo entero largo (long), esta función no retorna datos.

En algunas aplicaciones se requiere que la secuencia de números generados por la función random(...) difiera en cada ejecución, entonces es recomendable utilizar la función randomSeed(...) para inicializar el generador de números aleatorios con una entrada aleatoria como puede ser leer un pin analógico desconectado con la función analogRead(...).

El código ejemplo 6.4 muestra la forma de generar secuencias aleatorias diferentes cada 10 mseg por cada vez que se ejecute el sketch. Esto se realiza leyendo ruido del pin0, si éste se encuentra desconectado; otra forma de hacer secuencias aleatorias diferentes entre cada ejecución es de la siguiente forma: randomSeed(random(432)).

Las señales aleatorias son muy importantes para algunas áreas de la robótica, por ejemplo cuando se trata de identificar los parámetros del modelo dinámico de un robot (identificación paramétrica), por medio del algoritmo de mínimos cuadrados, se utilizan señales de excitación persistente, donde las señales aleatorias son parte principal en el proceso de convergencia paramétrica.

Alfaomega Arduino. Aplicaciones en Robótica y Mecatrónica

FERNANDO REYES CORTÉS • JAIME CID MONJARAZ

€ Código ejemplo 6.4

Generación de números aleatorios en cada ejecución.

```
void setup() {
    Serial.begin(9600);//abre puerto serial en 9600 Baudios.
}

void loop(){
    long num_aleatorio;
    //Cuando el pin 0 no tiene conectada una entrada, entonces tiene ruido.
    randomSeed(analogRead(0));//lee ruido estático en el pin0.
    num_aleatorio=random(432);//genera un número aleatorio entre 0 y 431.
    Serial.println(num_aleatorio);//exhibe el resultado en el monitor serial.
    delay(10);
}
```

Cuando se requiere que la secuencia de números aleatorios se repitan exactamente igual en cada ejecución, entonces se utiliza la función **randomSeed(...)** con un número fijo antes de empezar la generación de la secuencia aleatoria, por ejemplo **randomSeed(256)**.



6.3.9 Funciones para generar y detener tonos

Arduino tiene algunas funciones que pueden ser utilizadas como alarmas acústicas de alerta, debido a que pueden generar una señal cuadrada con diferentes frecuencias audibles y conectarse a bocinas o altavoces.

tone(pin, frecuencia, du)

La función **tone**(**pin**, **frecuencia**, **du**) genera una onda cuadrada de **frecuencia** específica en un **pin** con 50 % de ciclo de trabajo. Sólo se genera un tono en dicho pin cada vez que se ejecute esta función, con el cambio de **frecuencia** se genera otro tono.

Argumentos de entrada: **pin** de tipo unsigned int, indica el número de pin donde se genera el tono, la **frecuencia** en Hertz es un tipo de dato unsigned int, cuyo rango se encuentra comprendido entre 20 Hz y 20Khz, ya que corresponden a las frecuencias audibles por el ser humano (con las tarjetas Arduino no es posible generar tonos por debajo de 31 Hz). El argumento **du** es opcional y es del tipo unsigned long e indica la duración del tono en milisegundos. La función **tone(...)** no retorna datos.

Las aplicaciones más comunes de **tone(...)** es que una vez programado el pin puede conectarse a un zumbador piezoeléctrico o un altavoz (bocina) para generar tonos (alarmas). Debe tomar en cuenta que el uso de la función **tone(...)** interferirá con las salidas PWM de los pins 3 y 11.

Cuando el argumento **du** no se especifica, la onda cuadrada continúa hasta que se llame a la función **noTone(...)**. En este caso, la función **tone(...)** toma la siguientes sintaxis:

tone(pin, frecuencia)

Para detener la generación de ondas cuadradas generadas por **tone(...)** se utiliza la siguiente función:

∞ noTone(pin)

El argumento **pin** es del tipo unsigned int e indica el número de pin sobre el cual se desea detener la generación de onda cuadrada de tonos. Si en dicho **pin** aún no se ha generado algún tono, entonces la ejecución de **noTone(...)** quedará sin efecto. Esta función no retorna datos.

shiftOut(dataPin, clockPin, bitOrder, value)

El desplazamiento de un dato de longitud de un byte definido en el argumento value se envía bit a bit en el número de pin descrito en dataPin, el orden de la rotación o desplazamiento de los bits puede ser programado para iniciar con el bit más significativo (el bit que se encuentra en el extremo derecho de value) o también con el bit menos significativo (en el extremo izquierdo), cada bit de value es colocado en el pin especificado por dataPin, después de una transición de alto hacia bajo en el pin clockPin, el bit rotado ya se encuentra disponible para ser leído o procesado por algún dispositivo electrónico o interface. La función shiftOut(...) no retorna ningún tipo de datos.

En la tabla 6.9 presenta la descripción de los argumentos de entrada dataPin, clockPin, bitOrder y value que contiene la función shiftOut(...).

Cuando se emplean dispositivos electrónicos como interfaces en las tarjetas Arduino, cuya transición de reloj lo realizan en subida (de LOW hacia HIGH), entonces hay que asegurarse que **clockPin** se encuentre en bajo (LOW) antes de llamar a la función **shiftOut(...)**; lo anterior se puede lograr con la función **digitalWrite**(clockPin, LOW).

Tabla 6.9 Argumentos de entrada de la función shiftOut(dataPin, clockPin, bitOrder, value)

| Argumento de entrada | Descripción | |
|----------------------|---|--|
| dataPin | Tipo de dato entero (int) e indica el número de pin en el cual cada bit será rotado (configurado este pin previamente como puerto de salida a través de la función pinMode(pin, OUTPUT)). | |
| clockPin | Indica el pin que conmuta una vez que dataPin tiene el valor correcto. | |
| | Define el orden de rotación de los bits a desplazar, puede ser: primero el más significativo MSBFIRST o primero el menos significativo LSBFIRST. | |
| ${ m bitOrder}$ | MSBFIRST (most significant bit first). | |
| | LSBFIRST (least significant bit first). | |
| | Ambas constantes MSBFIRST y LSBFIRST han sido definidas en el sistema Arduino, y por lo tanto no requieren declaración. | |
| value | Es un dato del tipo byte (0 a 255) y representa el número a desplazar. | |

shiftIn(dataPin, clockPin, bitOrder) **○○**

Recibe el desplazamiento o rotación bit a bit de un dato (value) cuya longitud es de un byte; el desplazamiento de los bits que recibe puede ser configurado en dos formas posibles: una de ellas consiste en recibir primero el bit más significativo, otra opción es iniciar con el bit menos significativo. Por cada bit que ya está listo para ser leído en la línea de datos, el clockPin es puesto en estado alto (HIGH), esto sirve como protocolo para comunicación entre dos sistemas digitales. Una vez que el bit es leído por alguna interface, entonces clockPin es puesto en bajo (LOW).

Cuando se tienen interfaces electrónicas que trabajen con transiciones de subida (ciclos de reloj con el flanco de subida: de LOW hacia HIGH), se requiere asegurarse que el **clockPin** esté puesto en bajo (LOW) antes de utilizar la función **shiftIn(...)**, lo anterior se logra con la función **digitalWrite**(clockPin, LOW).

La tabla 6.10 muestra la descripción de los argumentos de entrada **dataPin**, **clockPin** y **bitOrder** para la función **shiftIn**(...).

Tabla 6.10 Argumentos de entrada de la función shiftIn(dataPin, clockPin, bitOrder)

| Argumento de entrada | Descripción | |
|----------------------|---|--|
| dataPin | Tipo de dato entero (int) e indica el número de pin en el cual cada bit será recibido (configurado este pin previamente como puerto de entrada a través de la función pinMode(pin, INPUT)). | |
| clockPin | Es el número de pin que conmuta cada vez que se lee dataPin . | |
| | Define el orden de rotación de los bits a leer, puede ser: primero el más significativo MSBFIRST o primero el menos significativo LSBFIRST. | |
| bitOrder | MSBFIRST (most significant bit first). | |
| | LSBFIRST (least significant bit first). | |
| | Ambas constantes MSBFIRST y LSBFIRST han sido definidas en el sistema Arduino, y por lo tanto no requieren declaración. | |

La función **shiftIn**(...) retorna el valor leído del puerto **dataPin** y corresponde a un número de tipo byte.



En varias aplicaciones de instrumentación, automatización, robótica y mecatrónica es útil medir el ancho o duración de un pulso, este es el caso de la función **pulseIn(pin, value, timeout)** que cronometriza la duración de una señal que permanece en estado alto o bajo. En la terminal de puertos especificada por el argumento de entrada **pin** se lee un pulso en estado alto (HIGH) o bajo (LOW) determinado por **value**, el argumento **timeout** es opcional y contiene el número de microsegundos en espera para leer el pulso. La tabla 6.11 describe los argumentos de entrada de esta función.

La función **pulseIn(pin, value, timeout)** retorna la duración del pulso en microsegundos o cero si no hay pulso antes del valor indicado en **timeout**. Debido a que el argumento **timeout** es opcional, entonces la función **pulseIn(...)** adquiere la siguiente sintaxis:

| pulseIn(pin, value) |
|---------------------|
|---------------------|

Tabla 6.11 Argumentos de entrada de la función pulseIn(pin, value, timeout)

| Argumento de entrada | Descripción |
|----------------------|---|
| pin | Tipo de dato entero (int) e indica el número de pin en el cual se leerá el pulso (configurado previamente como puerto de entrada a través de la función pinMode(pin, INPUT)). |
| value | Tipo de dato entero (int), contiene la clase de pulso a leer HIGH o LOW. |
| timeout | Tipo de dato entero largo sin signo (unsigned long), es un argumento opcional e indica el número de microsegundos para esperar a leer el pulso. Cuando no se especifica este argumento, su valor por defecto es un segundo. |

En este caso, el valor por defecto para timeout es de un segundo y esto se lleva a cabo de manera interna en el código de la función pulseIn(...). El código ejemplo 6.5 muestra la forma de utilizar a la función pulseIn(...).

Código ejemplo 6.5

```
función pulseIn(...).
```

```
int pin = 7;//número de pin donde se leerá el pulso.
//variable para registrar el ancho del pulso.
unsigned long duracion;
//Función de configuración.
void setup(){
    Serial.begin(9600); //velocidad de comunicación serial en 9600 Baudios.
    pinMode(pin, INPUT);//programa puerto digital como entrada.
}
//Lazo principal de programación.
void loop(){}
    duracion= pulseIn(pin, HIGH);//contabiliza duración del pulso.
    Serial.print("Duración del pulso en milisegundos=");
    Serial.print(duracion);
    delay(10);
```



6.3.10 Funciones para procesar bits y bytes

La manipulación individual de bits y bytes en variables de cualquier tipo se puede realizar a través de las siguientes funciones:

∞ lowByte(x))

Extrae el byte menos significativo, es decir, el primer byte que se encuentra en el extremo izquierdo del argumento \mathbf{x} , el cual representa una variable de cualquier tipo. Retorna un dato de tipo byte.

∞ highByte(x))

Extrae el byte más significativo (el último byte que se encuentra en el extremo derecho) de una variable o dato, el argumento de entrada está dado por \mathbf{x} y representa el valor de una variable de cualquier tipo. Esta función retorna un dato del tipo byte.

bitRaed(x, n)

Lee un bit de una variable \mathbf{x} de cualquier tipo, \mathbf{n} es un entero positivo que indica la posición del bit a leer, empezando en cero, que corresponde al bit menos significativo, es decir, el primer bit que se encuentra en el extremo derecho de la variable \mathbf{x} . Retorna el valor del bit siendo los posibles valores 0 o 1.

∞ bitWrite(x, n, b)

Escribe un bit en una variable numérica, donde el argumento \mathbf{x} es una variable de cualquier tipo, \mathbf{n} es un entero positivo que indica la posición del bit a escribir, empezando en cero, que corresponde al bit menos significativo, es decir, el primer bit que se encuentra en el extremo derecho de la variable \mathbf{x} y \mathbf{b} el valor del bit (0 o 1) a escribir en \mathbf{x} . Esta función no retorna datos.

∞ bitSet(x, n)

Escribe un bit con valor 1 en una variable \mathbf{x} de cualquier tipo, \mathbf{n} es un entero positivo que indica la

Alfaomega Arduino. Aplicaciones en Robótica y Mecatrónica

FERNANDO REYES CORTÉS • JAIME CID MONJARAZ

posición del bit a escribir, empezando en cero, que corresponde al bit menos significativo, es decir, el primer bit que se encuentra en el extremo derecho de la variable \mathbf{x} .

bitClear(x,n)



Limpia (escribe) un bit con valor 0 en la variable \mathbf{x} que puede ser de cualquier tipo, \mathbf{n} es un entero positivo, indica la posición del bit a escribir, el bit0 inicia en la primera posición que corresponde al bit menos significativo, ubicado en el extremo derecho de la variable \mathbf{x} .

La función bitClear(x,n) no retorna ningún tipo de datos.



6.3.11 Serial

Las tarjetas Arduino contienen módulos UART o USART para realizar comunicación serial con computadoras u otros dispositivos electrónicos. Se encuentran disponibles los pins digitales 0 y 1 para recepción (Rx) y transmisión (Tx), respectivamente, así como puerto USB para utilizarse como enlace con la computadora. Por lo tanto, al utilizar las funciones serial quedan inactivos los pins 0 y 1 como puertos digitales entrada/salida.

El modelo Mega tiene adicionalmente tres puertos seriales: uno de ellos corresponde al serial1 ubicado en los pins 19 (Rx) y 18 (Tx), otro puerto denominado serial2 está disponible en los pins 17 (Rx) y 16 (Tx), y el tercer puerto serial3 en los pines 15 (Rx) y 14 (Tx). Para utilizar estos pins en comunicación serial con una computadora se requiere un adaptador USB adicional a serie, ya que no están conectados al adaptador USB-Serie de la tarjeta Arduino Mega.

Para usarlos con dispositivos serie externos TTL, se necesita conectar el pin Tx al pin Rx del dispositivo y Rx al pin Tx del mismo dispositivo, además unir las tierras GDN de cada sistema (no conectar estos pins directamente a un puerto serie RS232, ya que opera a ± 12 V y puede dañar la tarjeta electrónica.)

Serial.begin(rate)



Esta función establece la velocidad de datos en bits por segundo (Baudios) para la transmisión de datos en forma serial, el argumento de entrada **rate** es de tipo entero largo (long) y el valor típico para comunicarse con una computadora es de 9600 Baudios. Sin embargo, otros valores de velocidad también pueden ser: 300, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600 o 115200. Esta función no retorna datos.

El modelo Arduino Mega contiene cuatro puertos seriales y se utilizan con las siguientes funciones: Serial.begin(rate), Serial1.begin(rate), Serial2.begin(rate) y Serial3.begin(rate).



Desactiva la comunicación serie, permitiendo que los pins Rx y Tx puedan ser usados como entradas o salidas digitales. Para reactivar la comunicación serie, se utiliza la función **Serial.begin()** y por lo tanto los pins digitales entrada/salida quedan bloqueados. Esta función no retorna datos.

Para la tarjeta Arduino Mega se utilizan las siguientes funciones: **Serial1.end()**, **Serial2.end()** y **Serial3.end()**.



Retorna el número de bytes (caracteres) disponibles para ser leídos por el puerto serie. Esto se refiere a los datos ya recibidos en el buffer de recepción del puerto serial, el cual tiene una capacidad de 128 bytes. Esta función no tiene argumentos de entrada. Para el modelo Arduino Mega se emplean las siguientes funciones: Serial1.available(), Serial2.available() y Serial3.available().



Lee los datos entrantes del puerto serie y retorna el primer byte disponible recibido por el puerto serie. Si no hay datos disponibles devuelve el valor -1. Esta función no tiene argumentos de entrada. En el modelo Arduino Mega se utilizan las siguientes funciones **Serial1.read()**, **Serial2.read()** y **Serial3.read()**.

El código ejemplo 6.6 muestra la programación necesaria para recibir datos desde el ambiente de programación Arduino instalado en una computadora personal y la tarjeta electrónica Arduino. En la rutina de configuración setup() se establece la velocidad de transmisión/recepción serial en 9600 Baudios; observe que en el lazo principal de programación loop() se utiliza la instrucción if(...){...} para detectar por medio de la función Serial.available() si hay datos disponibles a ser leídos en el buffer del puerto serial. Si el resultado que retorna es mayor que cero, entonces por medio de la función Serial.read() se lee el dato y se registra en la variables de tipo entero dato_read, este mismo dato se trasmite al monitor serial del ambiente de programación en formato decimal por medio de la función Serial.println(...) para desplegar la información que contiene.

€ Código ejemplo 6.6

Recepción de datos por comunicación serial.

```
int dato_read;//para registrar lecturas del puerto serial.
//Función de configuración.
void setup() {
    Serial.begin(9600);//abre puerto serial en 9600 Baudios.
}
void loop() {//Lazo principal de programación.
    //recibe datos solamente cuando están disponibles en el buffer.
    if (Serial.available() > 0) {
        dato_read=Serial.read();//lee el byte entrante desde la computadora.
        /*Se envía a la computadora para ser desplegado en el monitor serial del ambiente de programación Arduino.*/
        Serial.print("Dato que se ha transmitido desde la tarjeta Arduino: " );
        Serial.println(dato_read, DEC);//despliega dato en formato decimal.
    }
}
```

Serial.readBytes(buffer, num_char)



Lee un número determinado de caracteres (indicado por **num_char**) del puerto serial y los almacena en el argumento de entrada **buffer**, el cual corresponde a un registro o arreglo de datos del tipo char o byte, es decir son del tipo **char buffer**[num_char] o byte buffer[num_char].

La función Serial.readBytes(...) finaliza o termina su ejecución cuando la longitud de bytes en (num_char) ha sido leído o el valor de tiempo muerto de espera (time out) ha sido alcanzado (un segundo).

La función **Serial.readBytes(...)** retorna el número de caracteres colocados en el argumento de entrada **buffer**. Un 0 significa que ningún dato válido fue encontrado.

Serial.readBytesUntil(caracter_fin, buffer, num_char)



Lee un número de caracteres del puerto serial especificado en **num_char** y los almacena en el arreglo **buffer**; esta función finaliza si el carácter de terminación **carcater_fin** es encontrado, la longitud de caracteres **num_char** ha sido leída o el tiempo muerto (time out) ha sido alcanzado (un segundo). El argumento de entrada **buffer** es un arreglo de tipo char o byte (ers decir, **char buffer**[num_char] o byte buffer[num_char]).

Esta función retorna el número de caracteres leídos del puerto serial; un valor 0 significa que ningún dato válido fue encontrado.



Habilita el tiempo en espera (en milisegundos) de un dato serial cuando se utilizan las funciones Serial.readBytes(...) o Serial.readBytesUntil. El valor por defecto es 1000 milisegundos (un segundo). El argumento de entrada time es del tipo entero largo (long) e indica la duración del tiempo de espera en milisegundos. Esta función no retorna datos.

Serial.find(cadena)

Esta función lee datos del tipo char en el búfer serial, hasta que la **cadena** de caracteres sea encontrada; retorna un valor booleano verdadero si la **cadena** fue encontrada, un valor falso retorna para el caso cuando ha pasado suficiente tiempo (time out) y no fue encontrada dicha cadena.

Serial.findUntil(cadena, terminar)

Lee datos del tipo char en el búfer serial, hasta que la **cadena** o los caracteres del argumento **terminar** sean encontrados. Retorna un valor booleano verdadero si **cadena** fue encontrada, falso si ha pasado suficiente tiempo (time out) y no fue encontrada.

Serial.flush()

Limpia o vacía el búfer de entrada de datos seriales. Esto significa que cualquier llamada a Serial.read() o Serial.available() devolverá sólo los datos recibidos después de la llamada más reciente a Serial.flush(). Esta función no tiene parámetros de entrada, ni retorna datos.

Para el modelo Arduino Mega se emplean las siguientes funciones: **Serial1.flush()**, **Serial2.flush()** y **Serial3.flush()**.

Serial.parseFloat()

Retorna el primer número de punto flotante válido del buffer serial. Caracteres que no son dígitos o signo menos se omiten. La función **Serial.parseFloat()** es terminada por el primer carácter que no es un número de punto flotante. Esta función no tiene argumento de entrada.

Serial.Int()



Busca el próximo entero válido en la cadena serial entrante del flujo de datos serial, retorna dicho número entero. Si ningún entero válido es encontrado en un intervalo de un segundo (ajustable a través de la función **Serial.setTimeout(...)**), entonces retorna el valor 0. Esta función no tiene argumentos de entrada.

Serial.peek()



Retorna el primer byte (carácter) de los datos seriales entrantes sin removerlo del buffer serial interno o -1 si no hay datos disponibles. Esta función no tiene argumentos

En el modelo Mega también se encuentran disponibles las funciones Serial1.peek(), Serial2.peek() y Serial3.peek().

Serial.print(dato)



Escribe o imprime el argumento de entrada **dato** en el puerto serial como texto ASCII, el argumento **dato** puede ser de cualquier tipo de dato, inclusive cadena de texto. Cuando **dato** representa alguna variable numérica, entonces cada dígito se convierte en un carácter ASCII; las variables de tipo float se imprimen en forma de dígitos ASCII con dos decimales por defecto. Para las variables de tipo byte, se envían como un solo carácter entre comillas simples. Las cadenas de texto o de caracteres se envían tal cual enmarcadas entre dobles comillas. Por ejemplo:

Serial.print(78);//se imprime como "78".

Serial.print(1.23456);//se imprime como "1.23".

Serial.print(byte(78));//imprime la letra N, ya que su código ASCII es 78.

//Cuando se trata de un simple carácter o letra se emplea comillas.

Serial.print('N');// imprime un simple carácter: letra N.

//Para una cadena de texto o de caracteres se emplean dobles comillas.

Serial.print("Hola mundo.");//es una cadena de texto, imprime: Hola Mundo.

La función **Serial.print(dato)** retorna un dato de tipo entero largo (long), que corresponde al número de bytes escritos.



La función **Serial.print(dato, formato)** admite como segundo argumento de entrada la forma de desplegar a la información numérica: **formato** o la base del número a imprimir, los valores permitidos se presentan en la tabla 6.12:

El argumento opcional formato:



Especifica el número de la base para variables de tipo entero.



Número de fracciones para variables de tipo float o double.

Tabla 6.12 Tipos de formatos de la función Serial.print(dato,formato)

| Argumento opcional formato | Descripción | | |
|----------------------------|---|--|--|
| ВҮТЕ | Carácter en formato ASCII. Por ejemplo: Serial.print(78, BYTE);//imprime N. | | |
| BIN | Número binario o base 2. Por ejemplo: Serial.print(78, BIN);//imprime 1001110. | | |
| OCT | Número octal o base 8. Por ejemplo: Serial.print(78, OCT);//imprime 116. | | |
| DEC | Número decimal o base 10. Por ejemplo: Serial.print(78, DEC);//imprime 78. | | |
| HEX | Número hexadecimal o base 16. Por ejemplo: Serial.print(78, HEX);//imprime 4E. | | |

Para los números de tipo flotante (float y double), el argumento **formato** se especifica con el número de fracciones a imprimir. Por ejemplo:

Serial.println(8.123456, 0);imprime sin fracciones: 8.

Serial.println(8.123456, 1);//imprime una fracción: 8.1.

Serial.println(8.123456, 2);//imprime con dos fracciones: 8.12.

 $Serial.println(8.123456,3);//imprime\ tres\ fracciones:\ 8.123.$

Serial.println(8.123456, 4);imprime con cuatro fracciones: 8.1234.

Serial.println (8.123456,5); imprime cinco fracciones: 8.12345.

Para variables flotantes y doubles, el número de fracciones que despliega la función **Serial.prin(data, formato)** puede ser mayor a 255.

Serial.println(data)

Esta función trabaja de la misma forma que **Serial.print(data)**, con la diferencia que después de imprimir la información contenida en el argumento **data**, genera un retorno de carro y avance de línea automático (inserta una nueva línea).

Una forma equivalente de programación de la función Serial.println(data) es por medio de Serial.print(" $\ \$ "), la cual genera retorno de carro e inserta nueva línea.

La función **Serial.println(data)** también admite los mismos comandos indicados en la tabla 6.12 para imprimir información numérica.

La tabla 6.13 muestra los comandos de las funciones **Serial.print(dato)** y **Serial.println(dato)** para insertar tabulaciones, retornos de carro, líneas, etc.

Tabla 6.13 Tipos de comandos de la función Serial.print(dato,formato)

| Comando | Descripción |
|---------|--|
| "\t" | Tabulador (espacio horizontal), por ejemplo: Serial.print("\t");//genera un espacio horizontal. Serial.print("");//otra forma de generar espacio horizontal. |
| "\v" | Espacio vertical, por ejemplo: Serial.print("\v");//genera un espacio vertical. |
| "\r" | Retorno de carro, por ejemplo: Serial.print("\r");//realiza retorno de carro. |
| "\n" | Inserta una nueva línea, por ejemplo: Serial.print("\n");//genera una nueva línea. Serial.println("");//otra forma de generar una nueva línea. |

Serial.write(dato)

Esta función escribe datos binarios al puerto serie, los datos enviados pueden ser desde un simple

byte o series de bytes, es decir **dato** puede ser un simple byte o tipo char; también puede ser una cadena de caracteres enviada como una serie de bytes, pero también puede enviar un arreglo de cadena de texto; en este caso toma la siguiente sintaxis:



```
Serial.write(buffer, len)
```

El argumento de entrada **buffer** es un arreglo de datos tipo byte o char y **len** indica la longitud del arreglo, es decir: **char buffer**[len].

La función **Serial.write(...)** retorna el número de bytes escritos.



SerialEvent()

Cuando un dato serial está disponible (para capturar el dato serial se utiliza la función **Serial.Read()**), entonces esta función puede ser llamada con la siguiente sintaxis:

```
void serialEvent(){
//conjunto de sentencias en lenguaje C.
}

Sólo para el modelo Mega se puede utilizar:
void serialEvent1(){
//conjunto de sentencias en lenguaje C.
}

void serialEvent2(){
//conjunto de sentencias en lenguaje C.
}

void serialEvent3(){
//conjunto de sentencias en lenguaje C.
```

Nota: la función SerialEvent() no es compatible con los modelos Esplora, Leonardo y Micro

6.4 Resumen 187



Librerías estándar C

En el sitio Web de este libro están disponibles diversos ejemplos y aplicaciones de las librerías estándar del lenguaje C para tarjetas Arduino.



Librerías Arduino

En el sitio Web de la presente obra se encuentran disponibles ejemplos que ayudan a comprender mejor el uso y empleo de las librerías Arduino.



Funciones Arduino

Se ha preparado diversos ejemplos con las funciones Arduino para su mejor comprensión y aplicación. Estos ejemplos didácticos se encuentran disponibles en el sitio Web de este libro. Se invita al lector que los descargue, estudie y ejecute en los diversos modelos de tarjetas Arduino.



Interrupciones y aplicaciones

El uso de rutinas de servicio de interrupciones, manejo de interrupciones externas e internas son temas complejos que merecen ser abordados en detalle. En el sitio Web de la presente obra dentro del capítulo 12 "Temas selectos de Arduino" se ha preparado una sección de interrupciones con aplicaciones, reforzada con varios ejemplos que realizan control en tiempo real. Se invita al lector que descargue esta información para mejorar sus conocimientos de programación y control del sistema Arduino.

6.4 Resumen



en el proceso de compilación cuando se invoca a una función, sólo el código objeto de esa función forma parte del sketch. Sin embargo, para utilizar las funciones de las librerías necesariamente se requiere insertar en la cabecera del sketch la directiva #include<nombre_lib.h>, donde nombre_lib.h es un archivo que contiene la sintaxis (no el código) de las funciones que se encuentran en esa librería, por ejemplo para usar librerías estándar de entrada/salida se utiliza #include<stdio.h>, asignación dinámica de memoria para apuntadores #include<stdlib.h>, funciones matemáticas #include<math.h>, para manejo y manipulación de cadenas se emplea #include<string.h>, etc.

Varias funciones de las librerías en C han sido modificadas y adaptadas a las características del sistema Arduino, ya que éste carece de un sistema operativo y los recursos de las tarjetas electrónicas están orientados a la filosofía de los sistemas empotrados o (embedded systems).

Por otro lado, cada vez un número ascendente de librerías Arduino se están desarrollando para diversas aplicaciones y también requieren el uso de la directiva #include<arduino_lib.h> para que se puedan programar, por ejemplo: para el control de servomotores #include<Servo.h>, comunicación Wi-Fi #include<WiFi.h>, protocolos de internet #include<Ethernet.h>, tarjetas Robot #include<ArduinoRobot.h>, etc. Las aplicaciones en automatización, robótica y mecatrónica son innumerables y día a día el sistema Arduino se posiciona como uno de las mejores atractivos electrónicos para realizar aplicaciones educativas, industriales, comerciales y de investigación.

Debe tomarse en cuenta que un conjunto grande e importante de funciones Arduino se pueden utilizar sin la necesidad de utilizar alguna librería específica para realizar manipulación y procesamiento de datos de cualquier tipo, incluyendo la composición individual de bits y bytes, así como automatizar procesos y comunicar las tarjetas Arduino con una computadora, por lo que no se requiere la directiva #include <archivo.h>; el mismo compilador del ambiente de programación Arduino reconoce la sintaxis de esas funciones e inserta el código objeto correspondiente dentro del programa o sketch en el proceso de compilación.



6.5 Referencias selectas

La s librerías estándar del lenguaje C, librerías Arduino y sus funciones en combinación con todas las herramientas de programación en C y las características de las tarjetas Arduino como sistemas empotrados, son una herramienta eficiente y de alto desempeño para aplicaciones en ciencias exactas e ingeniería.

En el sitio Web http://arduino.cc puede consultarse tutoriales, documentos técnicos, aplicaciones

Alfaomega

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

FERNANDO REYES CORTÉS • JAIME CID MONJARAZ

de las librerías estándar en C y Arduino, así como sus funciones.



http://arduino.cc

Las siguientes direcciones electrónicas proporcionan información sobre las librerías estándar y Arduino:



http://www.nongnu.org/avr-libc/user-manual/modules.html



http://www.nongnu.org/avr-libc/user-manual/group_avr_stdlib.html

6.6 Problemas propuestos



La s funciones que componen las librerías estándar del lenguaje C y Arduino representan una potente herramienta de programación que permiten implementar aplicaciones de ingeniería robótica y mecatrónica.

A continuación se propone una serie de ejercicios con la finalidad de evaluar los conocimientos adquiridos por el lector sobre aspectos y empleo de dichas librerías.

6.6.1 Considere la siguiente declaración de variable del tipo flotante:

float x=M_PI;//asignación de constante de tipo flotante.

Realizar un sketch que contemple los siguientes casos de conversión de la variable \mathbf{x} a variables de los siguientes tipos:

- a) char usando la función char(x).
- b) int usando la función int(x).
- c) byte usando la función byte(x).
- d) word usando la función word(x).
- e) long usando la función long(x).
- Presente los resultados obtenidos en el monitor serial del ambiente de programación Arduino.
- Discuta y analice qué sucede con los cuatro bytes de la variable flotante en el proceso de conversión de cada caso.

6.6.2 Considere las variables tipo entero y double:

```
int j=12; double x;
```

Diseño un procedimiento para convertir el valor de la variable entera \mathbf{j} al tipo de variable double \mathbf{x} .

6.6.3 Suponga que tiene la siguiente unión de datos:

```
union comando{
    int i;//campo de tipo entero.
    char c;//campo de tipo char.
    long u;//campo de tipo entero largo.
    float x;//campo de tipo flotante.
    word w;//campo de tipo entero.
};
```

Considere las siguientes sentencias:

comando.x=M_PI;//asignación de una constante de tipo flotante.

comando.x=bitSet(x,31);//modifica el bit de la posición 31 de la variable x.

Tomando en cuenta las propiedades que tiene la **unión de datos**, analice y estudie el efecto que tienen las anteriores sentencias. Conteste a profundidad las siguientes interrogantes:

- a) ¿Cuál es el valor del campo tipo char?
- b) ¿Qué valor obtiene el campo de tipo int?
- c) ¿Analice cómo se modifica el campo de tipo long?
- d) ¿Cuál es el valor del campo flotante?
- e) Indique el valor que obtiene el valor de tipo word.
- 6.6.4 Realizar un sketch que utilice el canal 0 analógico para medir una onda señal senoidal de frecuencia 1 de un $\rm Hz$ y amplitud ± 3 $\rm V$.
 - a) ¿Cómo acoplará o acondicionará las señal senoidal al canal analógico?
 - b) ¿Qué valores tendrán los niveles de voltaje: -3 V, 0 V y 3 V?
 - c) ¿Qué tipo de referencia de voltaje utilizará?
 - d) ¿Cuál será el factor de conversión de voltaje, si el número de bits es 10?

Presente la lectura de la señal en el monitor serial del ambiente Arduino.

6.6.5 Transfiera el dato 893 de una tarjeta Arduino UNO (como transmisora) hacia otra tarjeta Arduino UNO (receptora) por medio de las instrucciones **shiftIn(...)**, **shiftOut(...)**. Realice la correspondiente conexión adecuada de pins y presente el resultado en el ambiente de programación Arduino.

```
#include <Servo.h>
#include <Stepper.h>
void setup(){
    Serial.begin(9600);
}
void loop(){
    robot_control();
}
```

- 7.1 Introducción
- 7.2 Motores de corriente directa
- 7.3 Motores a pasos
- 7.4 Resumen
- 7.5 Referencias selectas
- 7.6 Problemas propuestos

Competencias

Presentar los fundamentos básicos de los motores de corriente directa y motores a pasos en control en lazo abierto. Se describe las librerías Arduino para control de motores, así como desarrollo de programación mediante código propio disponible al lector.

Desarrollar habilidades en:

Motores de corriente directa.

Motorreductores.

Servos.

Motores a pasos.

Programación de motores de corriente directa por medio de librerías y código propio.

Programación de motores a pasos usando librerías y código propio.

7.1 Introducción 193

7.1 Introducción



En el concepto de variables de estado y ecuaciones diferenciales ordinarias de primer orden; particularmente el control de motores es relevante para sistemas mecatrónicos, robótica e ingeniería en general. A grandes rasgos podemos clasificar un sistema de control en lazo abierto y lazo cerrado como se muestra en la figura 7.1.

En el esquema de control en lazo cerrado la respuesta del motor y(t) (posición angular) se retroalimenta y se compara con un ángulo deseado o referencia r(t) para obtener la señal de error e(t) = r(t) - y(t), esta variable es procesada por una estructura de control u(t) para cumplir con el siguiente objetivo: $\lim_{t\to\infty} e(t) \to 0$. Es decir, que la posición del motor alcance el ángulo deseado, de esta forma el error de posición e(t) es cero, en otras palabras el control es exacto. Si esto se cumple, significa que desde el punto de vista técnico de control automático, el punto de equilibrio del sistema formado por el esquema o algoritmo de control y la planta (motor) es asintóticamente estable. Por lo tanto, se obtiene exactitud en el posicionamiento.

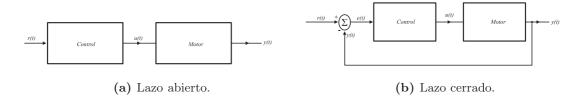


Figura 7.1 Control en lazo abierto y cerrado.

Sin embargo, los motores retroalimentados (en lazo cerrado) utilizan sensores (encoders de posición) para detectar el comportamiento de la respuesta o salida del motor; generalmente esos sensores no son baratos debido a sus características de resolución, salida digital, inmunidad al ruido, vibraciones y cambio de temperatura, así como peso ligero; además, los algoritmos de control en lazo cerrado son más complejos ya que su diseño involucra aspectos tales como teoría de sistemas dinámicos (lineales y no lineales) y estabilidad en el sentido de Lyapunov.

Los motores empleados en lazo cerrado se les denomina servomotores, los cuales son sistema electromecánico que convierte la energía eléctrica en movimiento mecánico (energía cinética y potencial), es decir son actuadores y también son considerados como transductores, debido a que transforman un tipo de energía (eléctrica) a otro tipo de energía (mecánica). Los servomotores se emplean en la construcción de los sistemas mecánicos que forman a los CNC's (máquinas de control numérico), robots manipuladores y sistemas mecatrónicos.

Servomotores



Servomotores son la evolución tecnológica del motor eléctrico que incorpora un sistema electrónico específico a su comportamiento dinámico conocido como servoamplificador y un sensor de posición con alta resolución, siendo generalmente el encoder.



Actualmente los servomotores representan la base del desarrollo que ha tenido la robótica y mecatrónica con una diversidad de aplicaciones en todos los sectores de la sociedad.

Por las características tecnológicas y ventajas que tienen, los servomotores son sistemas de precio elevado y en ocasiones no son fácil de conseguir, ya que dependen de la disponibilidad en stock o almacén del fabricante (proveedor). Una alterativa para bajar costos, tener buen compartimiento, eficiencia y usar sistemas de control simples para una diversidad de aplicaciones donde no se requiere emplear retroalimentación, son los denominados sistemas en lazo abierto.

La base tecnológica más empleada en sistemas de control de lazo abierto es la conversión de la señal de control en modulación de ancho de pulso PWM (por sus siglas en inglés: pulse width modulation); el rotor puede girar un determinado paso o ángulo fijo, de esta forma se realiza una correspondencia entre pulsos y ángulo de giro, no utilizan sensores y esquemas de control complejos. Evidentemente, con esta técnica no es posible lograr exactitud ideal en el posicionamiento como sería en el caso de los sistemas retroalimentados, ya que el posicionamiento es discreto o múltiplos del ángulo fijo de giro; para algunos motores es posible moverlos a medio pasos logrando movimiento más fino, otros más, por medio de sistemas de engranes se puede mejorar la resolución de movimiento del motor.

Servos

Los servos son un caso particular de servomotores con prestaciones más modestas, que no incorporan el servoamplificador, ni sensor de posición especializado (encoder) y cuya aplicación es control en lazo abierto, aceptando señales PWM.

La clasificación de motores utilizados en lazo abierto son los de corriente directa (DC o direct current), servos y motores a pasos (stepping motors o stepper motors).



7.2 Motores de corriente directa

os motores de corriente directa (DC por sus siglas en inglés *direct current*) corresponden al tipo de actuadores eléctricos más utilizado en ingeniería mecatrónica y robótica y en general se emplean en todos los aspectos de la vida cotidiana. Los motores eléctricos transmiten energía

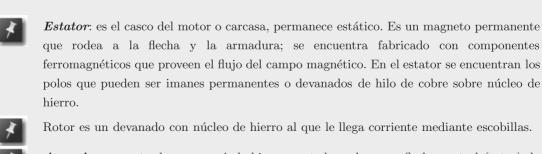
para producir movimiento en los sistemas mecánicos. El principio básico de los motores eléctricos se basa en la repulsión que ejercen los polos magnéticos con igual signo o polaridad entre estator y el rotor, el eje del rotor gira libremente entre los polos magnéticos norte-sur del imán permanente situado en la carcasa o estator del motor como se muestra en la figura 7.2.

Cuando la corriente eléctrica que suministra una fuente de voltaje V_{cc} circula por la bobina del rotor, se genera un campo electromagnético que interactúa con el campo magnético del imán permanente. Si los polos del estator y el rotor coinciden en polaridad, es decir son de signos iguales, entonces se produce un torque o par magnético por el rechazo de dichos polos, como consecuencia se genera movimiento giratorio alrededor del eje del rotor. Los componentes principales de los motores eléctricos son: armadura, conmutador, rotor (embobinado) y estator (la figura 7.2 muestra los componentes básicos).



Figura 7.2 Principio básico del motor eléctrico.

A continuación se describe los componentes básicos de los motores eléctricos:



Armadura: consta de una o más bobinas montadas sobre una flecha central (rotor); la corriente es conmutada a través de esas bobinas por medio de un conmutador. Como la fuerza de Lorentz impulsa a las bobinas, la fuerza de rotación (par o torque) se transmite a la flecha para que gire (fuerza contra-electromotriz). La ley de Lorentz establece que cuando un conductor transporta corriente y se encuentra en un campo magnético, entonces se genera una fuerza ortogonal (perpendicular) al flujo de corriente, produciendo el movimiento de rotación de un motor eléctrico.

Conmutador o colector: consta de dos platos divididos en la flecha o rotor, lo que proporciona potencia a la armadura de las bobinas, además de que están conectados a la fuente de alimentación.

El conmutador permite el cambio constante de polaridad de corriente en la bobina del rotor cada vez que completa media vuelta con la finalidad de que los polos norte y sur de la bobina del rotor coincidan respectivamente con los polos norte y sur del estator, de esta forma se genera una fuerza de repulsión debido a que los polos son iguales, generando movimiento rotacional por el torque o par magnético. Esto permite que el rotor se mantenga girando en forma permanente mientras se mantenga conectada la fuente de alimentación. Al proceso de alternar o conmutar el sentido de polaridad en la bobina se le denomina proceso de conmutación y representa el fundamento básico de los motores eléctricos. Algunos motores tienen escobillas (brush motors) para aumentar el flujo del campo magnético entre el estator y el rotor. Sin embargo, este tipo de motores incrementa la fricción y efectos térmicos. En contraste, los motores que no tienen escobillas (denominados brushless motors) tienen mejor desempeño.

Existen varios modelos y tipos de motores de corriente directa (ver figura 7.3), las especificaciones, características dependen del fabricante; sin embargo, de manera generalmente tienen tres cables: alimentación, referencia o tierra y señal de control. Para la mayoría de los motores, el cable de alimentación es de color rojo, para aquellos motores que no consumen demasiada corriente (300 mA) puede ser conectado al pin 5V de la tarjeta Arduino. El cable de referencia eléctrica o tierra es de co-



Figura 7.3 Motores de DC.

lor negro y la señal de control puede tener otro color y es conectado a un puerto digital o analógico. Cuando el motor requiere mucho más corriente que la que puede proporcionar el cable USB, entonces se requiere conectar el motor a una fuente de voltaje externa.



Figura 7.4 Componentes del servo.

En la figura 7.4 se muestran los componentes que forman un servo; consta de un pequeño motor de corriente continua que gira con alta velocidad y magnitud pequeña de par o torque, al rotor se le acopla mecánicamente un sistema de engranes para reducir la velocidad de giro y elevar el par alrededor del eje de giro. El sistema electrónico convierte la señal de radiocontrol (pulsos modulados con periodo de 20 ms) en voltaje continuo para que el rotor gire a la posición especificada.



7.2.1 Motor shield

El motor shield es una interface electrónica que permite controlar motores de DC a través de las tarjetas Arduino. Es un controlador de motores de puente completo dual que es soportado por el circuito integrado L298, el cual se emplea para cargas inductivas (bobinas) como relevadores, selenoides, motores de DC y a pasos.

Esta interfaz electrónica puede manejar dos motores de DC (controla velocidad y dirección de giro de cada motor por separado) o uno de pasos.

La etapa de alimentación se encuentra dividida para la etapa de electrónica digital y para manejar la potencia de los motores. Es conveniente colocar una fuente de alimentación externa (en el rango de 7V a 12V), ya que los motores generalmente exceden la cantidad de corriente que puede suministrar el puerto USB. El pin $V_{\rm in}$ se utiliza para conectar esta fuente de alimentación externa; sin embargo, debe tomarse en cuenta que $V_{\rm in}$ suministra energía al motor shield y a la tarjeta Arduino.

Cuando un motor requiere más de 9V, se recomienda separar las líneas de alimentación de la interfaz motor shiled y la tarjeta Arduino, es decir se necesita hacer que esta línea de alimentación sea dedicada al motor, esto se realiza dejando abierto el puente (jumper, colocado en el respaldo de la tarjeta de motor shield). El límite máximo voltaje en $V_{\rm in}$ es 18V y puede soportar hasta 2A por canal (máximo 4A por los dos canales).

El freno mecánico de cada motor se activa cuando en los pins A0 o A1 (del canal analógico de las tarjetas Arduino) detectan un Ampere (en realidad sensan 1.65V por cada Ampere: 1.65V/A). La tarjeta motor shield se instala directamente sobre las tarjetas Arduino, en la parte posterior del circuito impreso de motor shield contiene un conjunto de conectores tipo macho que se insertan sobre los conectores tipo hembra de las tarjetas Arduino.

La tabla 7.1 muestra las características principales de la tarjeta motor shield. Tiene dos canales (A y B) para controlar motores, el canal A contiene el puerto digital 12 (pin 12) para proporcionar el sentido de giro del motor y el pin 3 entrega la señal PWM. De manera análoga para el canal B, los pins 3 y 11, para dirección y señal PWM, respectivamente.

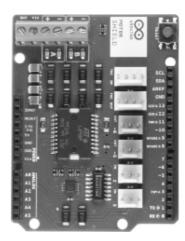


Figura 7.5 Motor shield.

Tabla 7.1 Características de motor shield.

| Voltaje | 5V a 12V |
|-------------------------------|---|
| Circuito | L298P |
| Motores | Dos motores de DC o uno a pasos |
| Dos canales (A y B) | Con fuente externa 2 A máximo por canal |
| Función (pins por canal) | Canal A Canal B |
| Dirección (positivo/negativo) | Pin 12 (Dir A) Pin 13 (Dir B) |
| PWM | Pin 3 (PWM A) Pin 11 (PWM B) |
| Freno (brake) | Pin 9 (break A) Pin 8 (break B) |
| Detección de corriente | $A0 \ ({\rm anal \acute{o}gico}) \ \ A1 \ ({\rm anal \acute{o}gico})$ |

♣ ♣ Ejemplo 7.1

Considere un motorreductor, realizar un algoritmo para girar al rotor en sentido positivo y negativo.

Solución

Controlar un motor de corriente directa en lazo abierto sin sistema de engranes es un proceso complicado, debido a que el rotor gira con alta velocidad y a pesar de que se le aplique un pequeño pulso al motor, el rotor girará varios grados antes de frenarse, debido al fenómeno inercial.

Por tal motivo, se requiere utilizar un sistema de engranes que permita disminuir la velocidad de giro y al mismo tiempo aumente la cantidad de torque. Un sistema con estas características se le denomina motorreductor.

Considere un motorreductor básico sin carga con un factor de reducción de velocidad 1:48 y bajo consumo de potencia; el voltaje en DC de alimentación se encuentra entre 3 y 6 V, 80 mA sin carga, tiene un torque de 0.008 Nm, y la velocidad nominal de giro es de 100 rpm y un peso 32 gramos; con carga puede demandar hasta 600 mA.

Las conexiones eléctricas del motorreductor a través de la interface motor shield (canal A) se muestran en la figura 7.6.



Motorreductor

Un motorreductor es un motor de corriente directa acoplado mecánicamente a un sistema de engranes, el rotor del motor se encuentra conectado al disco del engrane interior, mientras que la flecha del engrane funciona como rotor del motorreductor; existen en el mercado varias clases de motorreductores, entre ellos, los más básicos no cuentan con sistema electrónico, ni sensores de posición, es decir, estrictamente no son servos.

Observe que el motorreductor tiene dos cables, ambos conectados al canal A, uno de los cables está unido a la terminal etiquetada con signo — (puerto digital 12) y el otro al signo + (canal analógico A3). El puerto digital define el sentido de giro, en estado bajo (LOW) gira en sentido contrario a las manecillas del reloj y en estado alto (HIGH) es movimiento positivo, mientras que el canal analógico A3 proporciona energía para moverlo, el sistema de engranes se mueve lento; es posible realizar una calibración de modulación de ancho de pulso a grados.

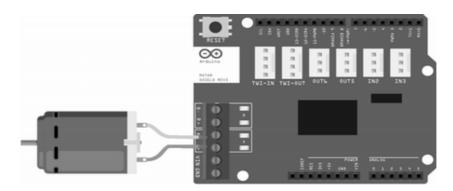


Figura 7.6 Motorreductor conectado a la interfaz motor shield.

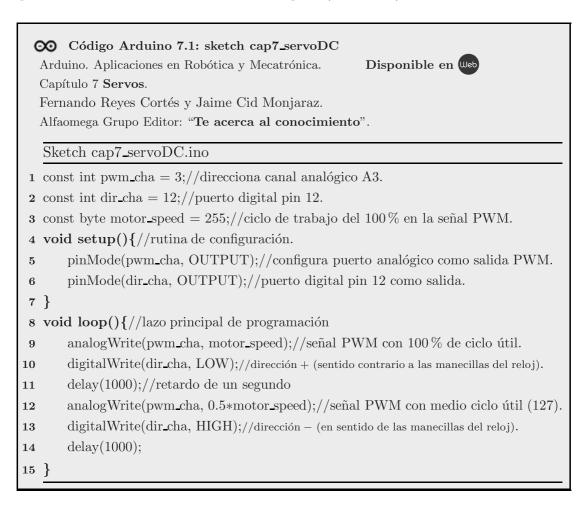
El cuadro de código Arduino 7.1 describe el sketch **cap7_servoDC** con la programación en lenguaje C para girar al rotor del motor en sentido positivo y negativo.

En la línea 1 se declara una constante de tipo entero **pwm_cha** para direccionar al canal 3 analógico suministrando señal tipo PWM, con un ciclo de trabajo del 100 % (255), la línea 2 la constante **dir_cha** direcciona al puerto digital pin 12 configurado como salida para darle cambio de giro a la rotación del motor.

La rutina de configuración **setup()** inicia en la línea 4, las constantes **pwm_cha** y **dir_cha** direccionan a los puertos analógico A3 y digital 12, respectivamente para ser programados como puertos de salida. El lazo principal de programación **loop()** del sketch empieza en la línea 8.

La línea 9 manda una señal PWM a unos de los cables del motor, mientras que el pin 12, definido por **dir_cha** define la dirección de giro del rotor (positivo si es LOW como se indica en la línea 9 y negativo para HIGH, tal y como se indica en la línea 13).

Cuando se aplica una señal PWM con el 100 % de ciclo útil, el motorreductor gira aproximadamente 5 grados y se detiene por un segundo; para movimientos negativos retrocede 2.5 grados debido a que se utiliza la mitad de ciclo útil **0.5*motor_speed** (ver línea 12).



Los pasos requeridos para ejecutar el sketch **cap7_servoDC** en cualquier modelo de las tarjetas Arduino son los que se describen en la sección 2.4 del capítulo 2 **Instalación y puesta a punto**

del sistema Arduino. Sin embargo, por comodidad al lector se presenta a continuación en forma resumida:

- Editar o en su caso cuando se indique descargar del sitio Web de este libro el código fuente en lenguaje C del sketch.
- En el menú **Herramientas** del ambiente de programación Arduino seleccionar modelo de tarjeta y velocidad de comunicación serial en 9600 Baudios.
- Compilar el sketch mediante el icono 🕢.
- Descargar el código de máquina a la tarjeta Arduino usando 👈
- Desplegar resultados con el monitor serial (activar con 🔑).

♣ ♣ Ejemplo 7.2

Controlar la rotación de un servo desde cualquier posición inicial hacia una referencia deseada q_d ; repetir este proceso en forma indeterminada.

Solución

Considere como caso de estudio el control de posición en lazo abierto de un servo, el cual se define como mover la posición del rotor desde cualquier posición inicial hacia un ángulo deseado o punto de referencia (dentro del rango de operación). Sea el modelo SRT2K5A, el cual es un tipo de servo económico (aproximadamente 12 dólares) y con disponibilidad en stock o almacén; generalmente, se utiliza en la construcción de pequeños sistemas mecatrónicos, robots móviles y también se emplea en aeromodelismo. En la figura 7.7 se muestra la modulación de ancho de pulso (PWM) para posicionar el rotor en el ángulo solicitado y en la tabla 7.2 se presentan las características técnicas del servo seleccionado.

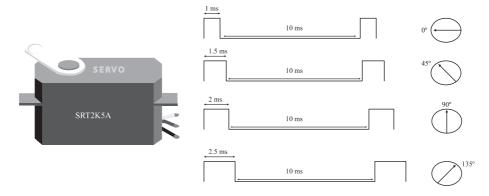


Figura 7.7 Señal PWM para el servo SRT2K5A.

El rango de operación de un servo representa el límite de giro que puede alcanzar el servo, esto depende de las características técnicas y mecánicas de fabricación, para el servo SRT2K5A se encuentra entre 0° a 135°. El control de un servo se realiza mediante pulsos cuadrados de voltaje cuya amplitud puede estar entre 4 a 8 V, el ángulo de rotación depende de la duración del pulso y varía de acuerdo al tipo de modelo y marca; dado un ancho de pulso definido por una duración de tiempo en ms en estado alto (HIGH), la electrónica interna del servo determina el ángulo que debe girar el rotor del servo. Por ejemplo, para el modelo SRT2K5A se emplean pulsos modulados con amplitud de 5 V, el periodo del pulso (tomando en cuenta la parte alta y baja) puede estar entre 11 ms a 12.5 ms, el ciclo útil del pulso o modulación se forma del estado alto (HIGH), cuya modulación puede ser de 1 ms, 1.5 ms, 2 ms y 2.5 ms (el resto del pulso en quedará en estado bajo o LOW durante 10 mseg) para posicionar al rotor en 0°, 45°, 90° y 135°, respectivamente (ver figura 7.7).

Tabla 7.2 Características del servo SRT2K5A.

| Voltaje | 4.8 a 6 V |
|--------------------|---|
| Torque | 0.023 Nm a 4.8 V y 0.025 Nm a 6 V |
| Peso | 45.5 g |
| Dimensiones | Altura=33 mm, largo=40 mm y ancho=13 mm |
| Velocidad de giro | 0.14 s/60° a 4.8 V y 0.16 s/60° a 6 V |
| Dirección | $1~\mathrm{a}~2~\mathrm{ms}$ |
| Rango de operación | 0° a 135° |

Desde el punto de vista de automatización, el problema de control en lazo abierto para un servo consiste en mover la posición del rotor desde cualquier punto inicial hacia un ángulo o referencia deseada.

Defínase la posición de casa en 0° y el ángulo deseado denotado por q_d , con unidades de medición en grados; el rango de operación del servo queda determinado por: $q_d \in [0, q_{\text{max}}]$, donde q_{max} representa el límite máximo de posicionamiento del servo. Para el modelo SRT2K5A el límite máximo está dador por: $q_{\text{max}} = 135^{\circ}$.

La conversión del ángulo en unidades de tiempo t (en milisegundos) para definir el ancho de pulso

(estado en alto o HIGH) de la señal PWM se establece a partir de la siguiente relación:

$$t = 1 + \frac{q_d}{q_{\text{max}}} \tag{7.1}$$

Debe tomarse en cuenta que $q_d < q_{\text{max}}$, es decir, el ángulo deseado no debe estar fuera del rango de operación.

El servo SRT2K5A tiene un conector eléctrico tipo hembra con 3 hilos; el pin 1 es de color negro (GND o tierra), pin 2 alimentación (color rojo) y pin 3 es la señal de control en forma de pulsos (cable de color amarillo, blanco o naranja).



Es importante indicar que el servo siempre sigue la señal de control (pulsos) y se debe aplicar a lo largo del tiempo t esta señal para que el servo alcance el ángulo deseado q_d y se mantenga en esa posición.



Cuando la señal de pulsos deja de aplicarse, entonces el servo queda en la última posición pero no ejerce ningún par para mantenerla (esta es una característica del lazo abierto). Es decir, cualquier fuerza externa aplicada sobre el rotor del servo lo moverá cambiando su posición.

El cuadro de código Arduino 7.2 presenta al sketch **cap7_servoA** con el algoritmo para posicionar al servo SRT2K5A un ángulo deseado q_d dentro de los límites físicos de operación (comprendidos entre 0° y 135°).

En este skecth no se utilizan librerías especiales para controlar al servo, más bien se desarrolla el procedimiento para control en lazo abierto. La interfaz electrónica entre el servo y la tarjeta Arduino UNO es el circuito motor shield (ver figura 7.8). La interface electrónica se inserta directamente en la parte superior de la tarjeta Arduino UNO.

En la línea 1 se declara **pin5** que identifica el puerto digital que envía la señal de control (pulsos PWM) al pin de entrada del servo (cable color amarillo o blanco).

En la línea 2 se define la variable **tiempo_angulo** para registrar el tiempo que se mantendrá en alto (HIGH) el pulso PWM; **qd** y **qmax** representan el ángulo deseado de posicionamiento y el límite máximo de movimiento del servo, respectivamente. La línea 5 contiene la subrutina de configuración **setup()** donde se programa el puerto digital **pin5 como** salida.

En la línea 8 se ubica la subrutina mueve_servo(tiempo) con el algoritmo para modular los pulsos

PWM entre 1 y 2.5 mseg en estado alto y 10 mseg en estado bajo (ver figura 7.7).

Esta modulación permite posicionar el rotor del servo en el ángulo deseado. Observe que para generar los tiempos de modulación de los pulsos se utiliza la función **delayMicroseconds** la cual permite mejor resolución en programar el ancho del pulso.

Cuando no se cuenta con las especificaciones o características de la modulación de los pulsos del servo, este procedimiento resulta útil para calibrar el desplazamiento angular en función del ancho de pulso.

A partir de la línea 16 inicia el lazo principal de programación del sketch; note que el servo es programado para moverse a la posición de casa (0°) mandando un tiempo de 1000 μ s (es decir, 1 ms) del pulso en estado alto, como se muestra en la línea 18.

El tiempo requerido para mantener en estado alto el pulso depende directamente del ángulo deseado q_d , la línea 21 implementa la ecuación 7.1 que determina la conversión de ángulo deseado q_d a tiempo de modulación del ancho de pulso PWM.

En la línea 22 se mueve al rotor al ángulo deseado, note que el argumento de entrada de la función $mueve_servo(...)$ es multiplicado por un factor de 1000, esto se debe a que la función utilizada en la modulación del pulso en alto trabaja con μ segundos.

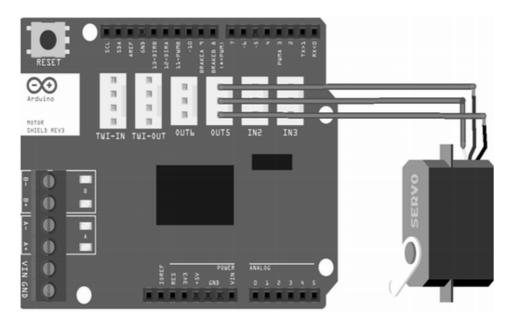


Figura 7.8 Conexiones eléctricas del servo con la interface electrónica motor shield.

```
○ Código Arduino 7.2: sketch cap7_servoA
  Arduino. Aplicaciones en Robótica y Mecatrónica.
                                                        Disponible en Web
  Capítulo 7 Servos.
  Fernando Reves Cortés y Jaime Cid Monjaraz.
  Alfaomega Grupo Editor: "Te acerca al conocimiento".
   Sketch cap7_servoA.ino
 1 int pin5 = 5;
 2 float tiempo_angulo;
 3 //Posición deseada y límite máximo en grados de movimiento para SRT2K5A.
 4 float qd=90.0, qmax=125.0;
 5 void setup(){//rutina de configuración.
       pinMode(pin5, OUTPUT);//puerto digital 5 configurado como salida.
 7 }
 8 void mueve_servo(int tiempo){//rutina para girar al servo.
9
       int i;
       for (i=0; i<100; i++){//aplicación continua de pulsos para alcanzar q_d.
10
          digitalWrite(pin5, HIGH);//pulso en estado alto.
11
          delayMicroseconds(tiempo);//tiempo del pulso en nivel alto (HIGH).
12
          digitalWrite(pin5, LOW);//pulso en nivel bajo (LOW).
13
          delayMicroseconds(10000);//tiempo de 10 ms (10000 \mus) en nivel bajo.
14
       }
15
16 }
17 void loop(){//lazo principal de programación.
       mueve_servo(1000);//posición de casa 0^{\circ} (1 mseg).
18
       //Ecuación que determina la conversión de tiempo de modulación en función del ángulo q_d.
19
       //t = 1 + \frac{q_d}{q_{\text{max}}}.
20
       tiempo_angulo=1+(qd/qmax);//cálculo del tiempo para alcanzar q_d.
21
22
       mueve_servo(1000*tiempo_angulo);//tiempo en microsegundos para alcanzar q_d.
23 }
24 //Para compilar, descargar y ejecutar este sketch, consulte la página 201.
```

El procedimiento para descargar el sketch **cap7_servoA** y su ejecución en las tarjetas Arduino se describe en la página 201.

Arduino tiene un conjunto de funciones para controlar motores de corriente directa usando la librería **Servo.h**, la descripción de esas funciones (sintaxis, tipos de argumentos de entrada, valores de retorno y tipo de clase) se detalla a continuación.



7.2.2 Librería Servo.h

Esta librería permite controlar (posicionar al rotor en un ángulo deseado) motores de corriente directa que generalmente contienen un sistema de engranaje para aumentar o bajar la velocidad de movimiento del rotor y amplificar el par o torque alrededor de la flecha o eje de giro. Rotaciones continuas sobre la flecha del rotor permiten variar la velocidad de movimiento. La librería **Servo.h** permite manejar hasta 12 motores DC en la mayoría de las tarjetas Arduino y 48 motores en el modelo Mega.

Para los modelos de tarjetas Arduino, con excepción del modelo Mega, el uso de la librería **Servo.h** deshabilita las funciones **analogWrite(...)** (pulsos PWM) en los pins 9 y 10. Sólo para el modelo Mega hasta 12 motores pueden ser utilizados sin interferir con las señales PWM, para 12 hasta 23 motores se tendrán que deshabilitar la funcionalidad PWM sobre los pins 11 y 12.



Asocia o establece el control de un motor de DC en un pin especificado por el argumento de entrada **pin**. La variable servo es un tipo de clase de tipo **Servo** y sirve para declarar funciones del usuario pertenecientes a este tipo de clase.

La función **servo.attach(...)** tiene parámetros opcionales para especificar el mínimo y máximo del ángulo de rotación. La sintaxis opcional es la que se describe a continuación:



donde **min** es el ancho del pulso en microsegundos correspondiente al mínimo ángulo de giro (0°) . El valor por default es 544 y el argumento de entrada **max** es el ancho del pulso en microsegundos que representa el giro máximo del motor (180°) ; por default el valor del argumento **max** es 2400.

Nota: para versiones Arduino 0016 o anteriores, la librería **Servo.h** maneja servos sólo en los pins 9 y 10.



Librería Servo.h

Ejemplos adicionales para controlar de servomotores se presentan en el sitio Web para aclarar conceptos y aplicaciones; así como, mejorar el entendimiento de la librería Servo.

Para ilustrar la forma de utilizar esta función, considere el siguiente código ejemplo 7.1:

∞ Código ejemplo 7.1

Funciones de la librería Servo.h

```
//Librería Servo.h en el header o cabecera del sketch.
#include <Servo.h>
Servo miservo;//crea objeto del tipo Servo.
void setup(){
   miservo.attach(9);
}
void loop() {
   miservo.write(70);//posición angular del servo en 70°.
   delay(15);
}
```

servo.attached()



Comprueba si la variable **servo** está unida o asociada a un pin del puerto digital. No tiene argumentos de entrada, se requiere que el tipo de variable sea de clase **Servo**. Retorna verdadero (true) si el **servo** está añadido a un pin, falso (false) en otro caso.

servo.detach()

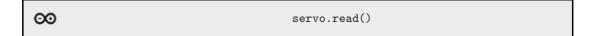


Quita o separa la variable **servo** del pin asociado al motor DC; si todas las variables son separadas, entonces los pins 9 y 10 pueden ser usadas para salidas de señales PWM con la función **analogWrite(...)**.

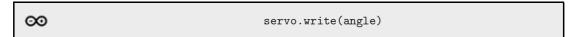
ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

Fernando Reyes Cortés • Jaime Cid Monjaraz

Alfaomega



Lee el ángulo actual del servo, es decir, el último valor utilizado por la función **write(...)**. No tiene argumentos de entrada, el tipo de clase de la variable debe ser **Servo**. Retorna el ángulo del servo comprendido entre 0° y 180°, suponiendo que 180° representa el límite máximo del rango de operación.



Escribe un valor o ángulo definido por el argumento de entrada (angle) al servo DC, entonces el rotor del motor girará al ángulo especificado. En una rotación continua del servo, un valor de 0 tendrá velocidad completa en una dirección, 180° (suponiendo que representa el límite máximo del rango de operación) tendrá velocidad completa en otra dirección y un valor de 90° no realiza movimiento; servo es una variable del tipo de clase **servo**. El argumento de entrada se encuentra comprendido entre 0° y 180° (si 180° es el límite máximo del rango de operación).



Escribe un valor en microsegundos (μ segundos) en el servo, el argumento de entrada **time** es de tipo entero (int). En servos estándar establecerá el ángulo de rotación del rotor del motor, un valor de 1000 corresponde a una vuelta completa.

Sin embargo, este valor depende del fabricante, ya que existe cierto grupo de servos que responden a valores de 700 a 2300. Servos con rotaciones continuas responderán de una forma similar a la función servo.write(...); servo es una variable de tipo Servo.

♣ ♣ Ejemplo 7.3

Controlar la rotación de un servo desde cualquier posición inicial hacia una referencia deseada q_d (el mismo planteamiento del ejemplo 7.2, pero ahora usando la librería **Servo.h**); repetir este proceso en forma indefinida.

Solución

Este ejemplo resuelve la misma problemática solicitada en el ejemplo 7.2, la diferencia se encuentra en utilizar funciones de la librería **Servo.h**. Por tal motivo, se selecciona el mismo modelo de servo SRT2K5A (ver tabla 7.2 y figura 7.8). En el cuadro de código Arduino 7.3 se presenta el sketch

cap7_servoB, en la línea 1 se declara en la cabecera del sketch la librería Servo.h usando la directiva #include "Servo.h", equivale a ser declarada como #include <Servo.h> (línea 2).

La línea 3 declara un objeto **miservo** de clase **Servo**. La configuración de **miservo** al pin 8 (línea 6) dentro de **setup()**. En el lazo principal **loop()** se programa movimiento positivo desde 0° hasta 135° (línea 9) y el retorno del rotor a la posición de casa o 0° se lleva a cabo en la línea 13.

El procedimiento para compilar el sketch **cap7_servoB**, decsraga de código y la ejecución en las tarjetas Arduino se describe en la página 201.

Código Arduino 7.3: sketch cap7_servoB Disponible en Web Arduino. Aplicaciones en Robótica y Mecatrónica. Capítulo 7 Servos. Fernando Reyes Cortés y Jaime Cid Monjaraz. Alfaomega Grupo Editor: "Te acerca al conocimiento". Sketch cap7_servoB.ino 1 #include "Servo.h" //librería para servos. 2 //#include <Servo.h>//sintaxis equivalente. 3 Servo miservo;//crea un objeto de clase servo, con un máximo de 8 servos. 4 int i;//pivote de la instrucción for(;;){...}. 5 void setup(){//subrutina de configuración miservo.attach(8);//configura pin 8 para control del servo. 7 } 8 void loop(){ for(i=0; i<135; i++){//movimiento positivo miservo.write(i);//posición angular del servo. 10 delay(15);11 12 for $(i=135; i>=1; i--){//movimiento negativo}$ 13 miservo.write(i);//posición angular del servo. 14 delay(15);**15** 16 17 } 18 //Para ejecutar este sketch en las tarjetas Arduino, consulte la página 201.



7.3 Motores a pasos

Os motores a pasos (steppers motors o stepping motors) son actuadores electromagnéticos rotacionales que convierten señales digitales (pulsos) en movimiento mecánico con desplazamiento angular fijo. La figura 7.9 presenta varios modelos de motores a pasos; la posición de rotación depende directamente del número de pulsos y la velocidad de movimiento está en función de la frecuencia de los pulsos.

Los motores de DC tienen movimiento rotacional en forma continua, mientras que los motores a pasos lo hacen como su nombre lo indica en forma discreta o a pasos, se mueven generalmente en forma más lenta que los motores DC, ya que hay un límite máximo para la velocidad a la que se pueden ir dando los pasos.



Figura 7.9 Motores a pasos.

Una característica típica de los motores a pasos es que cuando se encuentran detenidos, debido a la energía eléctrica que reciben las bobinas internas ejercen un freno mecánico que les impiden moverse, a diferencia de un motor de DC, que sólo tienen una bobina y que puede moverse cuando se encuentra estático.

Generalmente, los motores de pasos tienen cuatro bobinas que permiten avanzar o retroceder un pequeño ángulo de giro, llamado ángulo de paso, depende de las características mecánicas del motor y resolución (en radianes o grados). Típicamente los desplazamientos angulares pueden ser desde 0.75, 1.8, 3.6, 7.5, 15 y hasta 90 grados. La combinación de voltaje en las terminales de las bobinas define el movimiento del rotor del motor.

Los motores a pasos están formados por un rotor sobre el que se integran varios imanes permanentes y por un conjunto de bobinas en el estator. El eje del rotor se encuentra acoplado mecánicamente

sobre cojinetes que le permiten girar libremente. El motor a pasos tiene su funcionamiento en las fuerzas ejercidas entre los campos magnéticos de los imanes permanentes y los campos electromagnéticos que se generan cuando circula una corriente eléctrica; la bobina circular del estator se mantiene en una posición mecánica fija y en su interior se encuentra la bobina del rotor bajo la influencia de dichos campos generando un par electromagnético que propicia la rotación como se muestra en la figura 7.10.

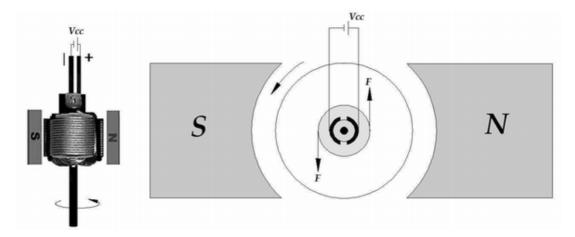


Figura 7.10 Funcionamiento básico de los motores a pasos.

Al polarizar la bobina del estator mediante voltaje DC se crean los polos N-S del campo magnético, el rotor tenderá a seguir el movimiento de dicho campo, es decir orientará sus polos N-S hacia los polos S-N del estator. Cuando el rotor logre esa alineación, entonces el estator conmuta la orientación de sus polos y el rotor se alineará con la nueva configuración. Este efecto se realiza de manera continua, consiguiendo el movimiento de rotación por desplazamiento angular de pasos, en otras palabras se transforma la energía eléctrica en energía mecánica (movimiento circular).

La conmutación o excitación de las bobinas es realizada por un dispositivo electrónico; existen en el mercado una gran cantidad de circuitos integrados para el control de motores a pasos en posición, velocidad y dirección (ver sitio Web del libro para listado de componentes electrónicos y ejemplos con diagramas esquemáticos para control electrónico de motores a pasos).

El número de pasos que gira el rotor cuando se realiza la conmutación o cambio de polaridad en las bobinas del estator se le denomina **ángulo de paso**. También se puede conseguir que el rotor gire medio paso mediante un control electrónico adecuado.

Los motores de pasos son fabricados para que trabajen en un ancho de banda específico (indicado en la hoja de datos del fabricante) y cuando se rebase este rango de frecuencias, se pierde la sincronización del motor.



Motores de pasos

Un motor de pasos es un dispositivo electromecánico cuyo rotor gira a través de movimientos angulares discretos que obedecen a una entrada digital (secuencia de pulsos), es decir, convierte la señal de pulsos digitales en movimiento mecánico discreto, por este motivo a un motor de pasos es considerado como un dispositivo discreto. Si la frecuencia del pulso se incrementa, entonces la velocidad de rotación aumenta. Los motores a pasos son adecuados particularmente para aquellas aplicaciones donde se utilizan señales de control como pulsos digitales más que señales analógicas.

Cuando una secuencia de pulsos digitales se aplica al motor, la flecha o rotor del motor a pasos se mueve en forma discreta y está directamente relacionada con la dirección o sentido de movimiento.

La velocidad de rotación depende de la frecuencia de la secuencia de pulsos, a mayor frecuencia, la magnitud de la velocidad se incrementa.

Algunas de las ventajas de los motores a pasos son las siguientes: el ángulo de rotación del motor es proporcional al pulso de entrada; el motor mantiene un torque cuando está detenido o estático (si el devanado está energizado). Estos dispositivos tienen buena respuesta de arranque.



Motores a pasos

En el sitio Web de esta obra se presentan una serie de algoritmos propios para el control de motores a pasos y aplicaciones en ingeniería robótica y mecatrónica.

La exactitud de movimiento de un paso se encuentra entre el 3 % al 5 % y este error no es acumulativo entre pasos consecutivos; los motores de pasos son considerados sistemas de control en lazo abierto, ya que no retroalimentan la respuesta, obedecen a pulsos de entrada, es decir no requieren de un encoder; se pueden obtener velocidades de movimiento muy bajas acoplando mecánicamente una carga al rotor.



7.3.1 Párametros importantes de los motores a pasos

Dentro de las aplicaciones en ingeniería hay que tomar en cuenta como parte de la selección de motores a pasos las siguientes características importantes:

- El ángulo de rotación de un motor a pasos es directamente proporcional al pulso de entrada.
- Número total de pasos N_p que puede dar un motor en un ciclo completo o 360 grados, se determina de la siguiente manera:

$$N_p = \frac{360}{\alpha} \text{ grados} \tag{7.2}$$

donde $\alpha \in \mathbb{R}_+$ es el ángulo de paso.

- Frecuencia de paso máximo es el número máximo de pasos por segundo que puede moverse el motor.
- Momento de inercia del rotor determina el efecto inercial en la dinámica del motor y proporciona información del tipo de carga que puede mover, así como la potencia del motor. Cuando el valor del momento de inercia es bajo, entonces el motor tiene una constante de tiempo pequeña, lo que produce respuesta rápida.
- Ángulo de paso es el desplazamiento angular que se mueve el rotor del motor por cada pulso de excitación. Por ejemplo la tabla 7.3 indica el número de desplazamientos que puede realizar el motor por cada ciclo completo (360°).
- Velocidad de movimiento es proporcional a la frecuencia de los pulsos.
- Par estático es el torque de freno del rotor cuando las bobinas del estator están desactivadas.
- Fricción de Coulomb: es una fuerza que se opone o resiste al movimiento, esta fuerza permanece constante con la velocidad de giro del rotor del motor a pasos.
- Fricción viscosa: es una fuerza que se opone al movimiento y es directamente proporcional a la velocidad de giro que tiene el rotor del motor a pasos.
- Driver: un sistema electrónico que convierte señales de pulsos a corriente para accionar un paso del motor.

| Ángulo de paso | Número de pasos |
|----------------|-----------------|
| 0.72° | 500 |
| 1.8° | 200 |
| 3.75° | 96 |
| 7.5° | 48 |
| 15° | 24 |
| 30° | 12 |
| 45° | 8 |
| 90° | 4 |

Tabla 7.3 Número total de pasos por cada 360°.

La tabla 7.3 muestra el número total de pasos que tiene que girar un motor para realizar una vuelta completa de 360°; dicho número total depende del **ángulo de paso**, el cual es una característica técnica del motor; entre más pequeño sea el ángulo de paso, el movimiento es mucho más fino o suave y más número de pasos se requerirán para completar un ciclo completo; por ejemplo si el paso es de 0.72°, entonces se requieren 500 pasos, note la diferencia cuando el paso es muy grande como 90° se requieren tan sólo 4 pasos para tener una vuelta completa. La figura 7.11 indica el movimiento positivo de rotación de un motor; se establece por convención en sentido contrario a las manecillas

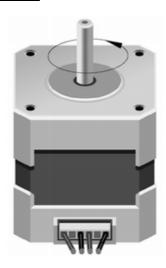


Figura 7.11 Rotación positiva.

del reloj y movimiento negativo en el sentido de las manecillas del reloj.



7.3.2 Motores a pasos con magneto permanente

El tipo de motor a pasos más común es el de imán o magneto permanente, tiene el rotor magnetizado que alterna polaridad (Norte-Sur) con las franjas paralelas magnéticas que se encuentran sobre la flecha del rotor (ver figura 7.12). El tamaño del paso angular depende del ancho de las franjas magnéticas (resolución angular). Hay motores a pasos de magneto permanente con 4 bobinas que pueden presentar hasta 8 terminales, 2 por cada bobina; las terminales se pueden unir interna o externamente dando lugar a dos tipos de motores a pasos: unipolares y los bipolares. Un motor

unipolar de 8 o 6 terminales se puede convertir en motor bipolar; pero lograr que un motor bipolar trabaje como unipolar se requiere separar sus bobinas internamente.

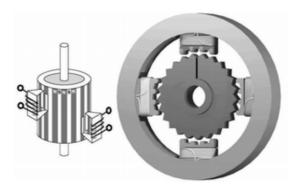


Figura 7.12 Motor a pasos con magneto permanente en el rotor, opera atracciónrepulsión entre el rotor y los electromagnetos (franjas paralelas) del estator.



7.3.3 Motores a pasos unipolares

Los motores de pasos unipolares están compuestos por dos y cuatro bobinas, generalmente vienen con 2 terminales por bobina; sin embargo, algunos motores de dos bobinas pueden presentar seis cables, cuatro de estos cables corresponden a los extremos de las bobinas y los otros dos hilos son derivaciones centrales o comunes de las bobinas como se ilustra en la figura 7.13. Como característica principal de este tipo de motores es que la corriente que circula por las bobinas lo hace en el mismo sentido, a diferencia de los motores bipolares. Los motores a pasos unipolares tienen varias formas de operación: paso simple, medio paso y paso doble.

Paso simple: en forma individual cada bobina del motor a pasos y por separado se activa o enciende, en este modo de operación no se adquiere suficiente torque o par en el rotor; la tabla 7.4 muestra las fases de activación de cada bobina y el correspondiente diagrama esquemático.

El modo de operación **paso doble** activa las bobinas en secuencias de pares, es decir, de dos en dos obteniendo un campo magnético más fuerte que en el caso de paso simple; la secuencia de encendido de las bobinas se ilustra en la tabla 7.5.

Medio paso combina los dos modos anteriores logrando que el rotor se moverá en pasos pequeños mejorando la exactitud para un trayecto completo de 360°. La tabla 7.6 presenta la secuencia de activación de las fases.

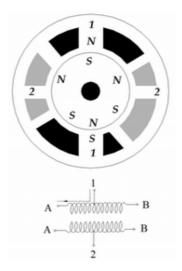


Figura 7.13 Motor a pasos unipolar.

Observe que para mover a un motor de pasos unipolar en el modo de medio paso se requieren más secuencias de activación de fases para las bobinas, con la ventaja de que el movimiento resultante es mucho más fino o suave respecto a los anteriores modos de operación.

Tabla 7.4 Paso simple para mover a un motor unipolar.

| Paso | A | В | С | D | Fase de activación |
|------|---|---|---|---|--------------------|
| 1 | 1 | 0 | 0 | 0 | |
| 2 | 0 | 1 | 0 | 0 | |
| 3 | 0 | 0 | 1 | 0 | m (m) m; |
| 4 | 0 | 0 | 0 | 1 | |

Tabla 7.5 Paso doble para mover a un motor unipolar.

| Paso | A | В | С | D | Fase de activación |
|------|---|---|---|---|--------------------|
| 1 | 1 | 1 | 0 | 0 | |
| 2 | 0 | 1 | 1 | 0 | |
| 3 | 0 | 0 | 1 | 1 | |
| 4 | 1 | 0 | 0 | 1 | |

Tabla 7.6 Medio paso para mover a un motor unipolar.

| Paso | A | В | С | D | Fase de activación |
|------|---|---|---|---|--------------------|
| 1 | 1 | 0 | 0 | 0 | |
| 2 | 1 | 1 | 0 | 0 | |
| 3 | 0 | 1 | 0 | 0 | |
| 4 | 0 | 1 | 1 | 0 | |
| 5 | 0 | 0 | 1 | 0 | |

| Tabla 7. | 7 (| Continua | ción | de | la | tabla | 7.6. | |
|----------|-----|----------|------|----|----|-------|------|--|
|----------|-----|----------|------|----|----|-------|------|--|

| 6 | 0 | 0 | 1 | 1 | |
|------|---|---|---|---|--------------------|
| Paso | A | В | С | D | Fase de activación |
| 7 | 0 | 0 | 0 | 1 | |

Como ejemplo de un motor a pasos unipolar, considere el modelo 28BYJ-48 de 5V DC con reductor de engranes que se muestra en la figura 7.14. Dentro de las características técnicas de este motor se encuentra que tiene una resistencia de 31 Ω por cada embobinado (del cable rojo a cualquier bobina), puede trabajar en secuencias de 8-step o 4-step. Para la secuencia 4-step, el ángulo de paso es de 11.25° , es decir puede dar 32 pasos por revolución ($\frac{360^{\circ}}{32} = 11.25^{\circ}$), mientras que en la secuencia de 8-step se obtiene un movimiento mucho más fino, ya que el ángulo de paso es de 5.625° , por lo que se requieren 64 pasos para completar una vuelta completa, es



Figura 7.14 Motor 28BYJ-48 a 5 V.

decir: $\frac{360^{\circ}}{64} = 5.625^{\circ}$. Este motor contiene un sistema de engranes con un radio de reducción de $\frac{1}{64}$ (el valor exacto es $\frac{1}{63.68395}$ o 63.68395:1), lo que significa que la flecha de salida del engrane toma 4096 pasos (64*64) en dar una vuelta completa para la secuencia 8-step y 2048=32*64 pasos en la secuencia 4-step. La frecuencia de salida sin carga es de 800 pulsos por segundo (pps) y la frecuencia de entrada sin carga puede ser hasta 500 pps.

El motor a pasos unipolar 28BYJ-48 es de 4 fases y tiene un conector con 5 hilos o cables (las derivaciones centrales de las bobinas están unidas) para interface que se identifican por colores de la siguiente manera: A (azul), B (rosa), C (amarillo), D (naranja) y E (rojo). La librería Arduino #include < Stepper.h > sólo corre en modo 4-step. El motor a pasos 28BYJ-48 se utiliza ampliamente en sistemas de aire acondicionado, ventiladores, control de fluidos en ductos o tuberías, floppys y discos duros, etc.

Tabla 7.8 Motor modelo 28BYJ-48.

| Voltaje | 5 VDC |
|---|--|
| Número de fases | 4 |
| Velocidad de giro: | $\frac{1}{64}$ |
| Distancia cubierta por paso | 5.625°/64 |
| Frecuencia | 100 Hz |
| Resistencia en DC | $50\Omega \pm 7\% \ (25\ ^{\circ}\text{C})$ |
| Frecuencia de entrada (estático) | >600 Hz |
| Frecuencia (dinámico) | >1000 Hz |
| Consumo de corriente | 160 mA por bobina (320 mA en modo 4-step), 250 mA estático y 200 mA en movimiento. |
| Torque para vencer la fricción estática | >0.034 Nm (Newton metro) |
| Torque de posicionamiento | >0.034 Nm |
| Torque de fricción | > 0.0588 a 0.1177 Nm |
| Torque del motor en estado dinámico | > 0.0294 Nm |
| Ruido | 35 dB (120 Hz, sin carga) |
| Peso | 0.04 Kg |

La terminal común o central de las bobinas del motor a pasos 28BYJ-48 se puede energizar a 5~V para cambiar el sentido de giro del rotor a la izquierda o derecha y producir el efecto de corriente de reversa sin tener un circuito electrónico que pueda manejar esta corriente como tal, por lo que el cable E identificado por el color rojo se conecta a 5~V, en este caso se utiliza el circuito UNL2003 (ver figura 7.15a).

Por otro lado, el circuito L293D puede manejar corriente de reversa para controlar el sentido de giro del rotor y por lo tanto no se requiere la conexión común para producir el cambio de giro en cada bobina (ver figura 7.15b).

Secuencia de pasos: el motor se mueve en respuesta a secuencia de campos electromagnéticos internos, cuando se activan las bobinas (en valor HIGH). La tabla 7.9 indica la secuencia de activación para 4-step y la tabla 7.10 para 8 step. En ambas tablas se contempla la terminal E en 5 V, por lo que se utiliza el circuito ULN2003. La secuencia 4-step utiliza la librería Arduino #include <Stepper.h>, en esta etapa dos de las cuatro bobinas se activan, y sólo una bobina cambia en cada paso. Observe que para la secuencia 8-step se inicia con una bobina encendida A, posteriormente AB, B, BC, C, CD, D y DA.

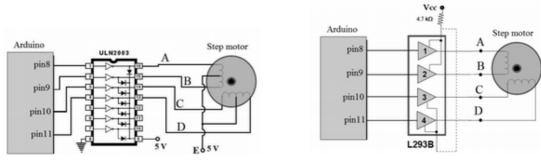
Tabla 7.9 Secuencias de activación 4-step del motor 28BYJ-48.

| | A (pin 1, azul) | | B (pin 2, rosa) | | C (pin 3, amarillo) | | D (pin 4, naranja) | | E (pin 5, rojo) | |
|------|-----------------|---------|-----------------|---------|---------------------|---------|--------------------|---------|-----------------------------------|--|
| Fase | Mov^+ | Mov^- | Mov^+ | Mov^- | Mov^+ | Mov^- | Mov^+ | Mov^- | Mov ⁺ Mov ⁻ | |
| 1 | Н | Н | L | L | L | L | Н | Н | 5 V | |
| 2 | Н | L | Н | L | L | Н | L | Н | 5 V | |
| 3 | L | L | Н | Н | Н | Н | L | L | 5 V | |
| 4 | L | Н | L | Н | Н | L | Н | L | 5 V | |

Tabla 7.10 Secuencias de activación 8-step del motor 28BYJ-48.

| | A (pin 1, azul) | | B (pin 2, rosa) | | C (pin 3, amarillo) | | D (pin 4, naranja) | | E (pin 5, rojo) | |
|------|-----------------|---------|------------------|------|---------------------|---------|--------------------|------------------|-----------------|---------|
| Fase | Mov^+ | Mov^- | Mov^+ | Mov- | Mov^+ | Mov^- | Mov^+ | Mov ⁻ | Mov^+ | Mov^- |
| 1 | Н | L | L | m L | L | L | L | Н | 5 | V |
| 2 | Н | L | Н | L | L | Н | L | Н | 5 V | |
| 3 | L | L | Н | L | L | Н | L | L | 5 | V |
| 4 | L | L | Η | Н | Н | Н | L | L | 5 | V |
| 5 | L | L | L | Н | Н | L | L | L | 5 | V |
| 6 | L | Н | L | Н | Н | L | Н | L | 5 | V |
| 7 | L | Н | L | L | L | L | Н | L | 5 | V |
| 8 | Н | Н | L | L | L | L | Н | Н | 5 | V |

En las tablas 7.9 y 7.10 la notación H significa estado alto o activo (HIGH) en los puertos digitales y L representa estado bajo o apagado (LOW). Por otro lado, la convención para determinar movimiento positivo corresponde al giro en sentido contrario a las manecillas del reloj, es decir antihorario y movimiento negativo en dirección a las manecillas del reloj (horario); en este contexto la notación Mov⁺ representa movimiento en sentido positivo, y Mov⁻ indica movimiento negativo.



(a) ULN2003: sin corriente de reversa

(b) L293B: corriente de reversa

Figura 7.15 Circuitos para manejar potencia y sentido de giro de motores a paso.

El motor a pasos unipolar modelo 28BYJ-48 a 5 V es muy fácil de adquirir, así como sus conexiones a las tarjetas Arduino resultan inmediatas. Sin embargo, la mayoría de los motores a pasos utiliza voltaje de alimentación mayor a 5 V.

Considere el caso de la figura 7.16 donde se ejemplifica el diagrama electrónico para un motor a pasos unipolar con 4 terminales; observe que la terminal COM (pin 9) del circuito ULN2003 se conecta a la fuente de alimentación del motor la cual puede estar en el rango de 5 V a 30 V.

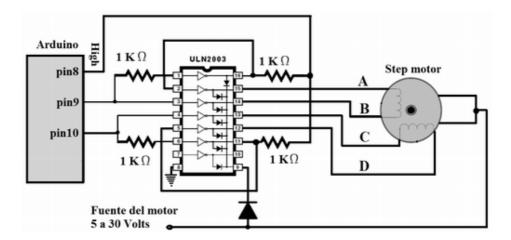


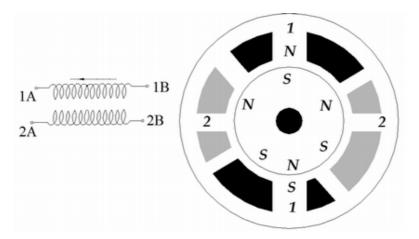
Figura 7.16 Motor a pasos unipolar de 4 terminales.



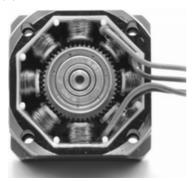
7.3.4 Motores a pasos bipolares

Los motores a pasos bipolares se forman por medio de 2 bobinas como se indica en la figura 7.15a, debido a que en los extremos de una bobina se pueden tener varias polaridades, entonces la corriente que circula por las bobinas cambia de sentido en función del voltaje.

Del motor salen cuatro cables como muestra la figura 7.15b, que corresponden a los extremos de cada bobina (dos cables por bobina).



(a) Motor bipolar con dos bobinas



(b) Parte interior del motor

Figura 7.17 Motor bipolar y estructura interna.

Los motores bipolares tienen aproximadamente un 30 % más torque que el caso de un motor unipolar del mismo volumen, pero su circuitería es mucho más compleja. Los circuitos electrónicos apropiados para poder enviar los patrones de señales de activación a las bobinas de un motor unipolar o bipolar son diferentes. En el caso de los motores bipolares, es necesario conectar a la tarjeta Arduino una

interfaz que acople la impedancia eléctrica y maneje la potencia adecuadamente, por lo general se realiza a través de un dispositivo denominado puente H, se requiere uno por cada bobina del motor, es decir dos "puentes H" iguales. Para los motores unipolares, lo más común es usar un conjunto de 8 transistores de tipo Darlington, normalmente encapsulados dentro de un mismo circuito integrado, como por ejemplo el dispositivo ULN2003 o ULN2004.



Para la mayoría de los motores a pasos se requiere mayor voltaje y corriente eléctrica de consumo que la que proporciona las tarjetas Arduino a través del puerto USB, así que generalmente se necesitará una fuente externa para alimentar los motores.



Existen varios modelos de motores a pasos y cada uno trabaja a un determinado voltaje DC; para saber cuál es el voltaje de trabajo del motor a utilizar se debe consultar la hoja de especificaciones del fabricante.



Algunos valores comunes de motores a paso son 5 V, 9V, 12V y 24V.

♣ ♣ Ejemplo 7.4

Controlar el giro de un motor de pasos unipolar para realizar vueltas completas de 360° contemplando movimientos positivos y negativos en la modalidad de 8-step.

Solución

Considere un motor a pasos unipolar modelo 28BYJ-48 a 5 V, el cual es un motor económico, fácil de adquirir y conectar a las tarjetas Arduino, no requiere fuente externa, la misma alimentación vía USB de la tarjeta electrónica es suficiente para suministrar energía a este motor. Para generar giros completos de 360° con movimientos suaves se utiliza la secuencias de activación 8-step tal y como se indica en la tabla 7.10; debido a que el motor seleccionado es unipolar, entonces el circuito apropiado para manejo de potencia y acoplamiento de señales es mediante arreglo de transistores Darlington: ULN2003 (ver sección de referencias para consultar proveedores de componentes electrónicos), la tarjeta Arduino utilizada es el modelo UNO, el diagrama electrónico que indica la forma de conectar el motor, circuito de potencia y tarjeta Arduino se muestra en la figura 7.18. Las componentes requeridas para realizar el experimento o práctica de controlar el giro del motor de pasos se muestran en la tabla 7.11 (es recomendable conseguir el circuito ULN2003 en circuito impreso con los diodos y resistencias ya incluidos).

Para abordar el problema planteado, el motor de pasos 28BYJ-48 se trabaja en la configuración 8-step (ver tabla 7.10 para secuencias de movimientos positivos y negativos), para alcanzar una vuelta

| Componente | Descripción | | | | |
|----------------------------------|---|--|--|--|--|
| Motor a pasos unipolar | Modelo 28BYJ-48 a 5 V | | | | |
| Circuito electrónico de potencia | Arreglo de transistores Darlington ULN2003 | | | | |
| Tarjeta Arduino | Modelo UNO | | | | |
| Diodos LEDs | Diversos colores para indicar secuencia de fase | | | | |
| Resistencias | 4 resistencias de 330 Ω a $\frac{1}{4}$ Watt | | | | |
| Cables de conexión | Varios tipo hembra/macho | | | | |

completa de 360° se requieren un total de 512 pasos tomando en cuenta el factor del reductor de engranaje del motor (64:1), es decir: (512=64*8).

Si se requiere posicionar el motor en algún ángulo deseado, se utiliza la siguiente regla proporcional:

grados =
$$\frac{512}{360^{\circ}}$$
 grados_deseados (7.3)

donde grados representa el ángulo de movimiento del rotor del motor, es decir, contiene el número de pasos o fases completas 8-step que tiene que girar el rotor hasta alcanzar el ángulo indicado; y grados_deseados es la referencia en donde se desea posicionar el movimiento del motor.

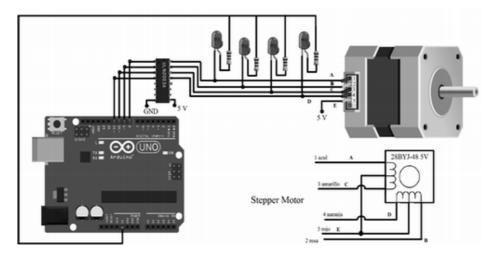


Figura 7.18 Diagrama electrónico del motor a pasos 28BYJ-48.

El cuadro de código Arduino 7.4 describe el sketch **cap7_StepMotor28B** cuya programación en

lenguaje C permite posicionar el motor en cualquier ángulo deseado (en realidad deben ser múltiplos de 0.703°).

Es ampliamente recomendable insertar una marca en el motor para identificar la posición de casa o posición 0° , y con respecto a esa referencia se realice la medición en grados del movimiento de rotación del rotor. En la línea 1 se declaran las variables que identifican los puertos digitales de la tarjeta Arduino conectadas al circuito ULN2003 y al motor a pasos. La línea 6 contiene el periodo (el inverso del periodo es la frecuencia) de activación de los pulsos aplicados a las bobinas, se ha seleccionado $1000~\mu$ segundos o un milisegundo (1 KHz), el ángulo de giro del rotor se registra en la variable **grados** y la posición deseada en **grados_deseados** (línea 9).

La línea 10 muestra la subrutina de configuración **setup()** donde se programan como puertos de salida los pins 8 al 11 y establece la velocidad de transmisión serial entre la tarjeta Arduino y la computadora donde se ubica el ambiente de programación Arduino.

En la línea 18 se utiliza la siguiente directiva:

$\#include < c: \arduino \Sketchs \cap7_movmotor \cap7_movmotor.ino >$

para incluir las subrutinas de giro positivo y negativo, indica la trayectoria o dirección del disco duro donde se encuentra el archivo **cap7_movmotor.ino** que contiene las subrutinas de movimiento (ver cuadro de código Arduino 7.5). Esta dirección electrónica es naturalmente un ejemplo y en tal caso es definida por el usuario.

El lazo principal **loop()** del sketch o programa **cap7_StepMotor28B** inicia a partir de la línea 19, el cálculo del ángulo que tiene que girar el rotor se encuentra en la línea 21 y queda registrado en la variable **grados**, la cual contiene el número de pasos que tiene que girar el rotor (fases completas 8-step). Observe como la variable **grados** trabaja como condición de salida en el primera instrucción **for(;;){...}** para posicionar al rotor en el ángulo indicado, la línea 23 contiene la rutina **giro_positivo(tiempo)** para posicionar al rotor en 360° con respecto a su posición de casa (posición 0°); para el caso de movimiento negativo se realiza con la función **giro_negativo** descrito en la línea 26. El lector puede asignar cualquier valor de ángulo en la variable **grados_deseados** (múltiplos de 0.703°) y comprobar que el motor alcanza esa posición. El argumento **tiempo** representa el periodo de los pulsos de activación para las bobinas del motor a pasos en las 8 fases (8-step). En este ejemplo, se utiliza 1000μ segundos o 1 milisegundo, significa para que el motor se mueva $0.703^{\circ} = \frac{360^{\circ}}{512}$ requiere completar una fase (8-step) en 8 milisegundos (cada paso de activación emplea un milisegundo a través de la función **delayMicroseconds(tiempo)**, ver cuadro de código Arduino 7.5), por lo que la velocidad de movimiento se obtiene como: $\frac{0.703^{\circ}}{8} = 87.89^{\circ}/\text{segundo}$.

En los cuadros de código Arduino 7.5, 7.5a, 7.5b y 7.5c se encuentran las rutinas de

```
€ Código Arduino 7.4: sketch cap7_StepMotor28B
  Arduino. Robótica y Mecatrónica.
  Capítulo 7 Servos.
  Fernando Reves Cortés y Jaime Cid Monjaraz.
  Alfaomega Grupo Editor: "Te acerca al conocimiento".
   Sketch cap7_StepMotor28B.ino
 1 int pin8 = 8;//Arduino: pin8->ULN2003 pin1, pin16->motor (pin1, A azul).
 2 int pin9 = 9;//Arduino: pin9->ULN2003 pin2, pin15->motor (pin2, B rosa).
 3 int pin10 = 10;//Arduino: pin10->ULN2003 pin3, pin14->motor (pin3, C amarillo).
 4 int pin11 = 11;//Arduino: pin11->ULN2003 pin4, pin13->motor (pin4, D naranja).
                //Arduino: pin 5V conectado al motor (pin5, color rojo).
 6 int tiempo=1000;//periodo de la señal de pulsos de entrada al motor.
 7 int i;//pivote para envío de la secuencia de activación 8-step.
8 int grados;//variable para registrar los grados de giro.
 9 float grados_deseados=360.0;//referencia deseada de giro.
10 void setup() {//subrutina de configuración.
       pinMode(pin8, OUTPUT);//pin8->pin1 ULN2003 pin16->A motor.
12
       pinMode(pin9, OUTPUT);//pin9->pin2 ULN2003 pin15->B motor.
13
       pinMode(pin10, OUTPUT);//pin10->pin3 ULN2003 pin14->C motor.
       pinMode(pin11, OUTPUT);//pin11->pin4 ULN2003 pin13->D motor.
14
       Serial.begin(9600);
15
16 }
17 //Incluye las subrutinas para movimiento positivo y negativo del rotor.
18 #include <c:\arduino\Sketchs\cap7_movmotor\cap7_movmotor.ino>
19 void loop(){//no se requiere ningún tipo de librería para motores a pasos.
       //Ángulo en grados del motor: grados=\frac{512}{360^{\circ}}grados_deseados.
20
       grados=(512.0/360.0)*grados_deseados;//ángulo de movimiento del motor.
21
       for (i=0; i<grados; i++){//gira el ángulo positivo.
22
23
         giro_positivo(tiempo);
24
       for (i=0; i < grados; i++) \{//gira el ángulo negativo.
25
         giro_negativo(tiempo);
26
27
  }//Para ejecutar este sketch consulte el procedimiento descrito en la página 201.
```

∞ Código Arduino 7.5: sketch cap7_movmotor

Arduino. Robótica y Mecatrónica.

Capítulo 7 Servos.

Fernando Reyes Cortés y Jaime Cid Monjaraz.

Alfaomega Grupo Editor: "Te acerca al conocimiento".

Sketch cap7_movmotor.ino

- 1 void giro_negativo(int tiempo){ //en dirección de las agujas del reloj (horario).
- 2 // Fase 1. Secuencia de activación de movimiento negativo (ver tabla 7.10).
- 3 digitalWrite(pin8, HIGH);//pin8->pin1 ULN2003 pin16->A motor.
- 4 digitalWrite(pin9, LOW);//pin9->pin1 ULN2003 pin15->B motor.
- 5 digitalWrite(pin10, LOW);//pin10->pin1 ULN2003 pin14->C motor.
- 6 digitalWrite(pin11, LOW);//pin11->pin1 ULN2003 pin13->D motor.
- 7 delayMicroseconds(tiempo);//frecuencia del pulso de activación.
- 8 // Fase 2.
- 9 digitalWrite(pin8, HIGH);//pin8->pin1 ULN2003 pin16->A motor.
- digitalWrite(pin9, HIGH);//pin9->pin2 ULN2003 pin15->B motor.
- digitalWrite(pin10, LOW);//pin10->pin3 ULN2003 pin14->C motor.
- digitalWrite(pin11, LOW);//pin11->pin4 ULN2003 pin13->D motor.
- delayMicroseconds(tiempo);//frecuencia del pulso de activación.
- 14 // Fase 3.
- digitalWrite(pin8, LOW);//pin8->pin1 ULN2003 pin16->A motor.
- digitalWrite(pin9, HIGH);//pin9->pin2 ULN2003 pin15->B motor.
- digitalWrite(pin10, LOW);//pin10->pin3 ULN2003 pin14->C motor.
- digitalWrite(pin11, LOW);//pin11->pin4 ULN2003 pin13->D motor.
- delayMicroseconds(tiempo);//frecuencia del pulso de activación.
- **20** // Fase 4.
- digitalWrite(pin8, LOW);//pin8->pin1 ULN2003 pin16->A motor.
- digitalWrite(pin9, HIGH);//pin9->pin2 ULN2003 pin15->B motor.
- digitalWrite(pin10, HIGH);//pin10->pin3 ULN2003 pin14->C motor.
- digitalWrite(pin11, LOW);//pin11->pin4 ULN2003 pin13->D motor.
- delayMicroseconds(tiempo);//frecuencia del pulso de activación.
- 26 // Fase 5.
- digitalWrite(pin8, LOW);//pin8->pin1 ULN2003 pin16->A motor.
- digitalWrite(pin9, LOW);//pin9->pin2 ULN2003 pin15->B motor.

Continúa código Arduino 7.5a: sketch cap7_movmotor Arduino. Robótica y Mecatrónica. Capítulo 7 Servos. Fernando Reyes Cortés y Jaime Cid Monjaraz. Alfaomega Grupo Editor: "Te acerca al conocimiento". Continuación del sketch cap7_movmotor.ino 29 digitalWrite(pin10, HIGH);//pin10->pin3 ULN2003 pin14->C motor. digitalWrite(pin11, LOW);//pin11->pin4 ULN2003 pin13->D motor. 30 delayMicroseconds(tiempo);//frecuencia del pulso de activación. 31 **32** // Fase 6. digitalWrite(pin8, LOW);//pin8->pin1 ULN2003 pin16->A motor. 33 digitalWrite(pin9, LOW);//pin9->pin2 ULN2003 pin15->B motor. 34 digitalWrite(pin10, HIGH);//pin10->pin3 ULN2003 pin14->C motor. 35 digitalWrite(pin11, HIGH);//pin11->pin4 ULN2003 pin13->D motor. 36 delayMicroseconds(tiempo);//frecuencia del pulso de activación. 37 38 // Fase 7. 39 digitalWrite(pin8, LOW);//pin8->pin1 ULN2003 pin16->A motor. digitalWrite(pin9, LOW);//pin9->pin2 ULN2003 pin15->B motor. 40 digitalWrite(pin10, LOW);//pin10->pin3 ULN2003 pin14->C motor. 41 digitalWrite(pin11, HIGH);//pin11->pin4 ULN2003 pin13->D motor. 42 delayMicroseconds(tiempo);//frecuencia del pulso de activación. 43 // Fase 8. 44 digitalWrite(pin8, HIGH);//pin8->pin1 ULN2003 pin16->A motor. 45 digitalWrite(pin9, LOW);//pin9->pin2 ULN2003 pin15->B motor. 46 47 digitalWrite(pin10, LOW);//pin10->pin3 ULN2003 pin14->C motor. digitalWrite(pin11, HIGH);//pin11->pin4 ULN2003 pin13->D motor. 48 delayMicroseconds(tiempo);//frecuencia del pulso de activación. 49 **50** } 51 void giro_positivo(int tiempo){//dirección anti horario. // Fase 1. Secuencia de activación de movimiento positivo (ver tabla 7.10). **52** digitalWrite(pin8, LOW);//A 53 digitalWrite(pin9, LOW);//B 54 digitalWrite(pin10, LOW);//C **55** 56 digitalWrite(pin11, HIGH);//D

○ Continúa código Arduino 7.5b: sketch cap7_movmotor

Arduino. Robótica y Mecatrónica.

Capítulo 7 Servos.

Fernando Reyes Cortés y Jaime Cid Monjaraz.

Alfaomega Grupo Editor: "Te acerca al conocimiento".

Continuación del sketch cap7_movmotor.ino

```
57
       delayMicroseconds(tiempo);//frecuencia del pulso de activación.
       // Fase 2.
58
       digitalWrite(pin8, LOW);//A
59
       digitalWrite(pin9, LOW);//B
60
       digitalWrite(pin10, HIGH);//C
61
       digitalWrite(pin11, HIGH);//D
62
       delayMicroseconds(tiempo);
63
       // Fase 3.
64
       digitalWrite(pin8, LOW);//A
65
       digitalWrite(pin9, LOW);//B
66
67
       digitalWrite(pin10, HIGH);//C
       digitalWrite(pin11, LOW);//D
68
       delayMicroseconds(tiempo);//frecuencia del pulso de activación.
69
70
       // Fase 4.
       digitalWrite(pin8, LOW);//A
71
       digitalWrite(pin9, HIGH);//B
72
73
       digitalWrite(pin10, HIGH);//C
74
       digitalWrite(pin11, LOW);//D
       delayMicroseconds(tiempo); //frecuencia del pulso de activación.
75
       // Fase 5.
76
       digitalWrite(pin8, LOW);//A
77
       digitalWrite(pin9, HIGH);//B
78
       digitalWrite(pin10, LOW);//C
79
       digitalWrite(pin11, LOW);//D
80
       delayMicroseconds(tiempo);//frecuencia del pulso de activación.
81
       // Fase 6.
82
```

digitalWrite(pin8, HIGH);//A digitalWrite(pin9, HIGH);//B

83

84

```
Continúa código Arduino 7.5c: sketch cap7_movmotor
  Arduino. Robótica y Mecatrónica.
  Capítulo 7 Servos.
  Fernando Reyes Cortés y Jaime Cid Monjaraz.
  Alfaomega Grupo Editor: "Te acerca al conocimiento".
   Continuación del sketch cap7-movmotor.ino
       digitalWrite(pin10, LOW);//C
85
       digitalWrite(pin11, LOW);//D
86
87
       delayMicroseconds(tiempo);//frecuencia del pulso de activación.
       // Fase 7.
88
       digitalWrite(pin8, HIGH);//A
89
       digitalWrite(pin9, LOW);//B
90
       digitalWrite(pin10, LOW);//C
91
       digitalWrite(pin11, LOW);//D
92
       delayMicroseconds(tiempo);//frecuencia del pulso de activación.
93
94
       digitalWrite(pin8, HIGH);//A Fase 8.
95
       digitalWrite(pin9, LOW);//B
       digitalWrite(pin10, LOW);//C
96
97
       digitalWrite(pin11, HIGH);//D
       delayMicroseconds(tiempo);//frecuencia del pulso de activación.
98
99 }
```

movimiento para controlar la dirección de giro en el sentido de las manecillas del reloj (horario) giro_negativo(tiempo) y para sentido contrario a las manecillas del reloj: giro_positivo(tiempo), ambas subrutinas se encuentran contenidas en el archivo cap7_movmotor.

El argumento de entrada **tiempo** determina el periodo de los pulsos de activación en cada etapa de la fase 8-step. La secuencia de activación de las bobinas ha sido programada de acuerdo a la tabla 7.10.



Para descargar el código de máquina y ejecutar el sketch cap7_StepMotor28B en las tarjetas Arduino consulte el procedimiento descrito en la página 201. Este sketch no requiere ninguna librería.

♣ ♣ Ejemplo 7.5

Controlar el giro de un motor de pasos unipolar a 180° (sentido contrario a las manecillas del reloj), tres segundos después que retorne a la posición inicial o de casa.

Utilice el modo de operación del motor en 4-step realizando subrutinas propias, es decir sin emplear librerías para motores a pasos.

Solución

Considere una vez más el motor a pasos unipolar modelo 28BYJ-48 a 5 V, para trabajar a este motor en modo de operación 4-step es necesario enviar a las bobinas pulsos de activación de acuerdo a la secuencia especificada en la tabla 7.9.

El diagrama electrónico que se utiliza para conectar la tarjeta Arduino UNO, motor a pasos y circuito ULN2003 es el que se observa en la figura 7.18. El motor parte de la posición de casa (0°) y gira en sentido positivo (dirección contraria a las manecillas del reloj) hasta alcanzar el ángulo de 180° , permanece en esta posición durante 3 segundos y posteriormente retorna a la posición de casa.

Para resolver el problema planteado se propone el algoritmo del cuadro de código Arduino 7.6, el cual describe el sketch **cap7_28BYJ4Step**. En la línea 1 se inicia la declaración de variables que definen los puertos a utilizarse como interfaz entre el motor y la tarjeta Arduino. La variable i de la línea 6 se emplea como pivote de las instrucciones **for(;;)**{...} que permiten alcanzar el ángulo deseado.

La línea 8 contiene la declaración de las variables **pasos** y **grados_deseados** para registrar la conversión de grados a número de pasos que tiene que girar el rotor para una determinada referencia.



En el modo de operación 4-step, el motor a pasos 28BYJ-48 gira 32 pasos internos para completar un giro de 360° (11.25° por cada paso).



Debido a que el rotor de este motor se encuentra acoplado mecánicamente a un sistema de engranes con factor de multiplicación 64:1, los pasos externos (en la flecha del sistema de engranes) para alcanzar una vuelta completa resultan de: 2048=64*32, este número de pasos externos debe ser dividido entre 4, ya que corresponde a la secuencia de fase 4-step para que el motor gire un paso externo en la flecha del sistema de engranes: $512=\frac{2048}{4}$ (lo que significa: $0.7031^{\circ}=\frac{360^{\circ}}{512}$ por paso).

Se ha seleccionado un periodo de 2000 μ segundos (2 milisegundos) en la señal de pulsos de las bobinas (ver línea 8), con este periodo una fase completa de 4 ciclos de activación requiere 8 milisegundos (2 milisegundos por fase de activación). Por lo tanto, la velocidad de movimiento del rotor es $\frac{0.7031^{\circ}}{8 \text{ milisegundos}} = 87.890^{\circ}/\text{segundos}$.

El número de pasos externos de la flecha del rotor en el sistema de engranes, pasos internos y las cuatro secuencias de activación de cada fase han sido declaradas en las líneas 9 y 11, respectivamente.

La configuración de los puertos digitales de salida se realiza en la rutina **setup()** (línea 12) y se establece la velocidad de transmisión serial en 9600 Baudios. La línea 19 tiene la directiva:

 $\#include < c: \arduino \Sketchs \cap7_movmotor4step \cap7_movmotor4step.ino >$

para incluir la ubicación en disco duro del archivo **cap7_movmotor4step.ino** con las subrutinas de movimiento (positivo y negativo en modo 4-step).

La dirección del archivo es definida por el usuario. A partir de la línea 20 inicia el lazo principal de programación loop() del sketch.



La conversión de **grados_deseados** a número de **pasos** que tiene que girar la flecha del sistema de engranes del motor para alcanzar esa referencia se lleva a cabo en la línea 21 a través de la siguiente ecuación:

pasos = num-pasos-motor_4step
$$\frac{\text{num-pasos-engrane}}{\text{num-fases-paso}} \frac{\text{grados-deseados}}{360.0^{\circ}}$$
 (7.4)



La rotación de movimiento positivo y negativo se lleva por medio de las funciones giro-positivo4step(tiempo) y giro-negativo4step(tiempo) (ver líneas 23 y 26, respectivamente).

En los cuadros de código Arduino 7.7 y 7.7a se describe el archivo que contiene las rutinas para movimiento positivo y negativo (cap7_movmotor4step.ino), de acuerdo a las especificaciones de secuencias de activación indicadas en la tabla 7.9.

Para descargar y ejecutar el sketch **cap7_28BYJ4Step** en las tarjetas Arduino consulte la página 201.

```
Código Arduino 7.6: sketch cap7_28BYJ4Step
  Arduino. Robótica y Mecatrónica.
  Capítulo 7 Servos.
  Fernando Reyes Cortés y Jaime Cid Monjaraz.
  Alfaomega Grupo Editor: "Te acerca al conocimiento".
   Sketch cap7_28BYJ4Step.ino
 1 int pin8 = 8;//Arduino: pin8->ULN2003 pin1, pin16->motor (pin1, A azul).
 \mathbf{2} int pin9 = 9;//Arduino: pin9->ULN2003 pin2, pin15->motor (pin2, B rosa).
 3 int pin10 = 10;//Arduino: pin10->ULN2003 pin3, pin14->motor (pin3, C amarillo).
 4 int pin11 = 11;//Arduino: pin11->ULN2003 pin4, pin13->motor (pin4, D naranja).
                //Arduino: pin 5V conectado al motor (pin5, color rojo).
 6 int i;//pivote para manejo de las instrucciones for(;;){...}.
 7 float pasos, grados_deseados=180.0;// número de pasos de giro y ángulo deseado.
 8 int tiempo=2000;//periodo de los pulsos de activación, 2000 μsegundos.
 9 #define num_pasos_engrane 64 //número de pasos externos en la flecha del engrane.
10 #define num_pasos_motor_4step 32 //número de pasos internos del motor.
11 #define num_fases_paso
                             4 //cuatro secuencias de activación por fase.
12 void setup(){ //subrutina de configuración.
       pinMode(pin8, OUTPUT);
13
       pinMode(pin9, OUTPUT);
14
       pinMode(pin10, OUTPUT);
15
       pinMode(pin11, OUTPUT);
16
       Serial.begin(9600);
17
18 }
19 #include <c:\arduino\Sketchs\cap7_movmotor4step\cap7_movmotor4step.ino>
20 void loop(){//lazo principal de programación del sketch.
21 pasos= (num_pasos_motor_4step*num_pasos_engrane/num_fases_paso)*(grados_deseados/360.0);
       for (i=0; i<pasos; i++){//giro para movimiento en sentido positivo.
22
         giro_positivo4step(tiempo);
23
24
       delay(3000);//3 segundos de pausa.
       for (i=0; i<pasos; i++){//giro para movimiento en sentido negativo.}
         giro_negativo4step(tiempo);
26
27
28 }
```

€ Código Arduino 7.7: sketch cap7_movmotor4step Arduino. Robótica y Mecatrónica.

Capítulo 7 Servos.

Fernando Reyes Cortés y Jaime Cid Monjaraz.

Alfaomega Grupo Editor: "Te acerca al conocimiento".

Sketch cap7_movmotor4step.ino

```
1 //En la dirección de las agujas del reloj horario (horario).
```

- 2 void giro_negativo4step(int tiempo){//secuencia de activación de la tabla 7.9.
- 3 // Fase 1.
- 4 digitalWrite(pin8, HIGH);//pin8->pin1 ULN2003 pin16->A motor.
- digitalWrite(pin9, LOW);//pin9->pin2 ULN2003 pin15->B motor. 5
- digitalWrite(pin10, LOW);//pin10->pin3 ULN2003 pin14->C motor. 6
- 7 digitalWrite(pin11, HIGH);//pin11->pin4 ULN2003 pin13->D motor.
- delayMicroseconds(tiempo);//frecuencia del pulso de activación. 8
- // Fase 2. 9
- 10 digitalWrite(pin8, LOW);//pin8->pin1 ULN2003 pin16->A motor.
- 11 digitalWrite(pin9, LOW);//pin9->pin2 ULN2003 pin15->B motor.
- digitalWrite(pin10, HIGH);//pin10->pin3 ULN2003 pin14->C motor. **12**
- digitalWrite(pin11, HIGH);//pin11->pin4 ULN2003 pin13->D motor. 13
- delayMicroseconds(tiempo);//frecuencia del pulso de activación. 14
- // Fase 3. 15
- digitalWrite(pin8, LOW);//pin8->pin1 ULN2003 pin16->A motor. 16
- 17 digitalWrite(pin9, HIGH);//pin9->pin2 ULN2003 pin15->B motor.
- digitalWrite(pin10, HIGH);//pin10->pin3 ULN2003 pin14->C motor. 18
- digitalWrite(pin11, LOW);//pin11->pin4 ULN2003 pin13->D motor. 19
- delayMicroseconds(tiempo);//frecuencia del pulso de activación. **2**0
- // Fase 4. 21
- digitalWrite(pin8, HIGH);//pin8->pin1 ULN2003 pin16->A motor. 22
- digitalWrite(pin9, HIGH);//pin9->pin2 ULN2003 pin15->B motor. 23
- digitalWrite(pin10, LOW);//pin10->pin3 ULN2003 pin14->C motor. 24
- digitalWrite(pin11, LOW);//pin11->pin4 ULN2003 pin13->D motor. 25
- delayMicroseconds(tiempo);//frecuencia del pulso de activación. 26
- 27 }

```
Arduino. Robótica y Mecatrónica.
  Capítulo 7 Servos.
  Fernando Reyes Cortés y Jaime Cid Monjaraz.
  Alfaomega Grupo Editor: "Te acerca al conocimiento".
   Continuación del sketch cap7_28BYJ4Step.ino
28 //En la dirección contraria a las agujas del reloj, movimiento anti horario.
29 void giro_positivo4step(int tiempo){//secuencia de activación de la tabla 7.9.
30
       // Fase 1.
       digitalWrite(pin8, HIGH);//pin8->pin1 ULN2003 pin16->A motor.
31
32
       digitalWrite(pin9, HIGH);//pin9->pin2 ULN2003 pin15->B motor.
       digitalWrite(pin10, LOW);//pin10->pin3 ULN2003 pin14->C motor.
33
       digitalWrite(pin11, LOW);//pin11->pin4 ULN2003 pin13->D motor.
34
       delayMicroseconds(tiempo);//frecuencia del pulso de activación.
35
36
       // Fase 2.
       digitalWrite(pin8, LOW);//pin8->pin1 ULN2003 pin16->A motor.
37
       digitalWrite(pin9, HIGH);//pin9->pin2 ULN2003 pin15->B motor.
38
       digitalWrite(pin10, HIGH);//pin10->pin3 ULN2003 pin14->C motor.
39
       digitalWrite(pin11, LOW);//pin11->pin4 ULN2003 pin13->D motor.
40
       delayMicroseconds(tiempo);
41
       // Fase 3.
42
       digitalWrite(pin8, LOW);//pin8->pin1 ULN2003 pin16->A motor.
43
       digitalWrite(pin9, LOW);//pin9->pin2 ULN2003 pin15->B motor.
44
45
       digitalWrite(pin10, HIGH);//pin10->pin3 ULN2003 pin14->C motor.
       digitalWrite(pin11, HIGH);//pin11->pin4 ULN2003 pin13->D motor.
46
47
       delayMicroseconds(tiempo);//frecuencia del pulso de activación.
       // Fase 4.
48
       digitalWrite(pin8, HIGH);//pin8->pin1 ULN2003 pin16->A motor.
49
       digitalWrite(pin9, LOW);//pin9->pin2 ULN2003 pin15->B motor.
50
51
       digitalWrite(pin10, LOW);//pin10->pin3 ULN2003 pin14->C motor.
52
       digitalWrite(pin11, HIGH);//pin11->pin4 ULN2003 pin13->D motor.
       delayMicroseconds(tiempo); //frecuencia del pulso de activación.
53
54 }
```

Continúa código Arduino 7.6a: sketch cap7_28BYJ4Step

Para ejecutar el sketch cap7_28BYJ4Step consulte el procedimiento descrito en la página 201.



7.3.5 Librería Stepper.h

La librería **Stepper.h** se utiliza en el modo **4-setp** para motores a pasos del tipo unipolar y se declara al inicio del sketch (header o cabecera) a través de la directiva **#include <Stepper.h**>, la cual contiene la sintaxis de un grupo de funciones para tarjetas Arduino.

El número de parámetros de entrada de esas funciones depende cómo se encuentre conectado el motor (dos o cuatro pins en las tarjetas Arduino).

La sintaxis de las funciones de esta librería se describe a continuación:

∞

Stepper(steps, pin1, pin2)

El parámetro de entrada **steps** indica el número de pasos en una revolución o vuelta completa de 360°, si se conoce el ángulo de paso, entonces se divide 360° entre ese ángulo.

Por ejemplo, si el ángulo de pasos es 7.5° , $steps = \frac{360^{\circ}}{7.5^{\circ}} = 48$; el parámetro steps es del tipo entero (int). Los parámetros de entrada pin1 y pin2 son datos de tipo entero (int) e indican qué pins del puerto digital de la tarjeta Arduino se encuentran conectadas al motor de pasos unipolar. Esta función retorna un objeto del tipo de clase **Stepper**.

Cuando el motor a pasos unipolar tiene conectado 4 pins a la tarjeta Arduino, entonces la función **Stepper(...)** tiene dos parámetros más opcionales con la siguiente sintaxis:

∞

Stepper(steps, pin1, pin2, pin3, pin4)

donde los parámetros de entrada **pin3** y **pin4** indican qué tipo de pins del puerto digital de la tarjeta Arduino se encuentran conectados al motor de pasos unipolar. Por ejemplo:

//Pins de la tarjeta Arduino conectados al motor de pasos. int pin8=8, pin9=9, pin10=10, pin11=11; Stepper motor_pasos;//tipo de clase Stepper. motor_pasos=Stepper(48,pin8,pin9, pin10, pin11);

Sobre la función clase **Stepper(...)** ver nota de corrección en la página 239.

setSpeed(rpms)



El parámetro de entrada **rpms** del tipo entero largo (long) establece la velocidad del motor en rotaciones por minuto; esta función no hace que el motor gire, únicamente especifica la velocidad de rotación cuando se utilice la función **step()**. La función **setSpeed(...)** no retorna ningún tipo de dato.

step(rpms)



Esta función hace que gire el motor un número específico de pasos determinado por el argumento de entrada **steps** a una velocidad especificada por la función **setSpeed(...)**.

La función **step(...)** es del tipo **blocking**, es decir que durante su ejecución esperará hasta que el motor finalice su movimiento, entonces el sketch continuará con la siguiente instrucción.

Es importante hacer notar el lector que si establece una velocidad de una revolución por minuto y el número de pasos del motor es 100, entonces la función **step(...)** podría tomar más de un minuto su ejecución.

El parámetro de entrada **steps** es del tipo entero, un valor positivo significa que el motor girará en el sentido contrario a las manecillas del reloj, en caso contrario el motor girará en el sentido de las manecillas del reloj.

♣ ♣ Ejemplo 7.6

Controlar el giro de un motor de pasos unipolar a 180° (sentido contrario a las manecillas del reloj), tres segundos después que retorne a la posición inicial.

Utilice el modo de operación del motor 4-step con la librería Stepper.h.

Solución

Para posicionar a un motor a pasos unipolar en modo 4-step desde la ubicación de 0° (posición de casa), se utiliza como caso práctico el motor 28BYJ-48 a 5 V; el cuadro de código Arduino 7.8 muestra la documentación del sketch **cap7_StepMotor4Step** con la programación específica para mover al motor a 180° y 3 segundos después regresarlo a la posición de casa.

Debido a que al motor a pasos 28BYJ-48 se trabaja en modo de operación 4-step se emplea la librería **Stepper.h** (ver línea 1 en el header del sketch).

Los pins del puerto digital de la tarjeta Arduino se encuentran declarados de la línea 2 a la 5, la variable **grados_deseados** registra la ubicación indicada (en este caso 180°) donde se desea posicionar al rotor del motor, así como la variable **pasos** para registrar el cálculo interno del número de pasos necesarios para alcanzar la posición deseada, la cual es definida por el usuario.

Para ejecutar el sketch cap7_StepMotor4Step consulte el procedimiento de la página 201.

○ Código Arduino 7.8: sketch cap7_StepMotor4Step Arduino. Aplicaciones en Robótica y Mecatrónica. Disponible en Web Capítulo 7 Servos. Fernando Reyes Cortés y Jaime Cid Monjaraz. Alfaomega Grupo Editor: "Te acerca al conocimiento". Sketch cap7_StepMotor4Step.ino 1 #include <Stepper.h>//librería para motores de pasos en modo 4-step. 2 int pin8 = 8;//Arduino: pin8->ULN2003 pin1, pin16->motor (pin1, A azul). 3 int pin9 = 9;//Arduino: pin9->ULN2003 pin2, pin15->motor (pin2, B rosa). 4 int pin10 = 10;//Arduino: pin10->ULN2003 pin3, pin14->motor (pin3, C amarillo). 5 int pin11 = 11;//Arduino: pin11->ULN2003 pin4, pin13->motor (pin4, D naranja). //Arduino: pin 5V conectado al motor (pin5, colo rojo). 7 float pasos, grados_deseados=180.0; 8 #define num_pasos 9 Stepper mimotorapasos(num_pasos, pin8, pin10, pin9, pin11);//protocolo de sintaxis. 10 void setup(){ //subrutina de configuración. Serial.begin(9600);//establece velocidad de comunicación serial en 9600 Baudios. 12 mimotorapasos.setSpeed(200);//velocidad de rotación por minuto. 13 } 14 void loop(){//lazo principal de programación del sketch. pasos=grados_deseados*2048.0/360;//conversión del ángulo deseado a pasos. 15 //Movimiento positivo en sentidos contrario a las manecillas del reloj. 16 mimotorapasos.step(-pasos);//sentido positivo: se mueve a la posición deseada. 17 delay(3000);//3 segundos de pausa. 18 //Movimiento negativo en el sentido de las manecillas del reloj. 19 20 mimotorapasos.step(pasos);//regresa a la posición inicial. 21 } 22 //Para ejecutar este sketch consulte el procedimiento descrito en la página 201.

El ángulo de paso del motor 28BYJ-48 en modo 4-step es de 11.25° por paso, es decir, se requieren 32 pasos internos para lograr 360° (sin tomar en cuenta el sistema de engranes), además recuerde que este motor tiene un sistema de engranaje acoplado mecánicamente al rotor con un factor de reducción de 64:1, es decir la flecha del sistema de engranes gira 64 pasos externos para dar una vuelta completa. Entonces, el número total de pasos para alcanzar un giro completo es: 2048=32*64. En la línea 8 se define el número de pasos (64) del sistema de engranaje del motor 28BYJ-48 para dar una vuelta completa de 360°. La tabla 7.11 muestra el listado de componentes necesarios para realizar esta práctica y el diagrama electrónico que realiza la interface entre la tarjeta Arduino, el motor a pasos y el circuito UNL2003 es el mismo que se describe en la figura 7.18.



En la línea 9 se realiza el protocolo de sintaxis para definir la rutina del usuario del tipo de clase **Stepper**:



Stepper mimotorapasos(num_pasos, pin8, pin10, pin9, pin11)

Para un correcto funcionamiento de la librería **Stepper.h** es necesario respetar el orden en que han sido declarados los argumentos en la función definida por el usuario **mimotorapa-**sos(num-pasos, pin8, pin10, pin9, pin11).



La corrección es por software, observe que **pin10** es el tercer argumento de entrada (de izquierda a derecha) y precede al argumento **pin9**, además no se requiere hacer ningún tipo de modificación en el cableado del circuito ULN2003, ni en el motor. En otras palabras, el diagrama electrónico de la figura 7.18 se mantiene sin cambios.

La rutina de configuración **setup()** se ubica en la línea 10 y establece la velocidad de comunicación serial en 9600 Baudios y la velocidad de rotación del rotor en 200 rpm. En la línea 14 inicia el lazo principal de programación del sketch.

La conversión del ángulo deseado a **pasos** necesarios para alcanzar la referencia indicada por **grados_deseados** se realiza en la línea 15; el usuario puede cambiar el valor del ángulo deseado de rotación alrededor del eje de giro del rotor y observar que efectivamente se posiciona el rotor en el ángulo programado.

La línea 17 emplea la función **mimotorapasos.step(-pasos)** mueve al rotor la cantidad de **pasos** necesarios para alcanzar la referencia deseada, el signo menos indica por convención que la dirección de giro es en sentido contrario a las manecillas del reloj (movimiento positivo).

Después de 3 segundos el rotor gira en sentido de las manecillas del reloj (movimiento negativo) para regresar a la posición de 0° utilizando el argumento **pasos** sin el signo negativo como se observa en la línea 20. Este proceso se repite de manera indefinida por el lazo del programa **loop()**.

Note que en contraste al cuadro de código Arduino 7.4 que describe la documentación técnica del sketch cap7_StepMotor28B del ejemplo 7.4 se requieren mucho más líneas de programación (ver cuadros de código Arduino 7.5, 7.5a, 7.5b y 7.5c para el modo de operación del motor en 8-step). Una de las ventajas de utilizar librerías como Stepper.h es la reducción de código de programación, ya que facilita el proceso de automatización. Sin embargo, siempre será conveniente saber cómo se realizan o implementan las funciones para poder acondicionarlas o adaptarlas a cualquier tipo de aplicación específica o particular.



Para descargar el código de máquina y ejecutar el sketch **cap7_StepMotor4Step** en las tarjetas Arduino consulte el procedimiento descrito en la página 201.



Ejemplos con motorreductores

En el sitio Web de este libro, se encuentran disponibles diversos ejemplos y aplicaciones con motorreductores. Programación en código fuente de lenguaje C para sistemas mecatrónicos.



Ejemplos con motores a pasos

En el sitio Web de la presente obra, se encuentran disponibles aplicaciones con motores a pasos. Ejemplos de programación en lenguaje C de sistemas mecatrónicos didácticos.



7.4 Resumen

Los servomotores son sistemas conformados por un motor de corriente directa, sistema electrónica o manejador (driver) y sensor para medir el movimiento rotacional, cuya información se emplea en retroalimentación para propósitos de control en lazo cerrado; una clase particular de servomotores son los denominados servos como son los motores de corriente directa con sistema de engranes (motorreductor) y motores a pasos; estos dispositivos son sistemas muy básicos en comparación con un servomotor y no incluyen sensor de posición; por lo tanto, se emplean en control de lazo abierto, el manejador electrónico está diseñado para aceptar señales PWM o entradas por modulación de ancho de pulso para mover al rotor un ángulo fijo, el movimiento lo hacen en forma discreta (a pasos fijos), la posición rotacional depende directamente del número de pulsos

7.4 Resumen 241

y la velocidad de giro está directamente relacionada con la frecuencia de los pulsos PWM. Los motorreductores tienen movimiento rotacional en forma continua, sin embargo al aceptar señales PWM giran de manera discreta de la misma forma que los motores a pasos.

Generalmente los motorreducotres carecen de un sistema electrónico y de sensores de posición, por lo que se requiere establecer un algoritmo que permita calibrar el movimiento rotacional o paso fijo de giro en función de la modulación del ancho de pulso. El acoplamiento electrónico con un sistema digital como las tarjetas Arduino se hace a través de un puente H como es la inteface motor shield.

La parte clave de control en lazo abierto para un servo es la conversión de pulsos digitales PWM en movimiento rotacional; el sistema electrónico del servo contiene un algoritmo que determina el giro de rotación del rotor en función del ancho de pulso de la señal PWM, con esta equivalencia no se requiere utilizar sensor de posición, la sección electrónica del servo resulta muy simple, económico y fácil de operar. Los servos tienen una amplia gama de aplicaciones en lazo abierto, por ejemplo: hornos de microondas, impresoras de matriz, graficadores, plotters, discos flexibles (floppys), bandas transportadoras, copiadoras, prototipos científicos, puertas automáticas, equipo e instrumental médico, etc.

Los motores de pasos tienen las siguientes características:

- Este tipo de motores responden a pulsos digitales de entrada, funcionando como sistemas de control en lazo abierto.
- El ángulo de rotación del motor es proporcional al pulso de entrada.
- Cuando el motor se encuentra estático, mantiene un torque alrededor del eje de giro (si las bobinas están energizadas).
- Dependiendo de las características del motor, la exactitud de posicionamiento y repetitibilidad de movimiento es del 5 % y este error no es acumulativo de un paso a otro.
- Excelente respuesta de movimiento: inicio, paro y reversa.
- La velocidad de rotación del motor a pasos (alrededor de la flecha o rotor) es proporcional a la frecuencia de los pulsos de entrada, de esta forma se puede obtener un amplio rango de velocidades de movimiento.
- Es posible lograr velocidades de movimiento muy bajas (lentas) cuando se acopla una carga directamente a la flecha.
- La interface electrónica para conectar un motor a pasos con las tarjetas Arduino es por medio del dispositivo ULN2003.

Los motorreducotres ofrecen mayor torque o par que los motores a pasos, sin embargo los motores

a pasos tienen mayor exactitud en aplicaciones en lazo abierto.

7.5 Referencias selectas



La bibliografía para servos (motorreducotres y motores a pasos) y sus aplicaciones en el área de ingeniería resulta muy extensa, no obstante se recomienda consultar los referencias contenidas en los siguientes enlaces electrónicos.



www.arduino.cc

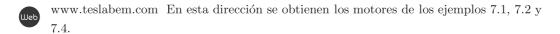


www.tigoe.net/pcomp/code/circuits/motors/

Consultas y dudas sobre aplicaciones de Arduino con motorreductores y motores ver el siguiente enlace:

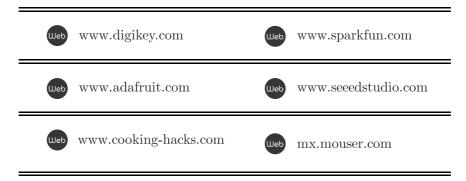


En los siguientes enlaces puede adquirir motorreductores, motores a pasos, interfaces electrónicas, accesorios, kits, etc.:





Interfaces y componentes electrónicos pueden ser obtenidos en:



7.6 Problemas propuestos



Los servos son dispositivos electromagnéticos que convierten pulsos digitales del tipo PWM en movimiento mecánico rotacional y que son operados en lazo abierto; a continuación se propone un conjunto de problemas para evaluar el grado de comprensión del lector sobre los conceptos y los diversos planteamientos presentados en este capítulo.

- 7.6.1 Considere un motorreductor como el utilizado en el ejemplo 7.1, desarrollar un sketch con la tarjeta Arduino UNO para que el motorreductor pueda realizar el seguimiento de las siguientes referencias deseadas q_d :
 - a) Sea una referencia variables en el tiempo de la forma $q_d = 120^{\circ} \text{ sen}(t)$ donde t es la variable tiempo cuya evolución se realiza en múltiplos de 100 mseg.
 - b) Movimientos periódicos desde $q_d=90^{\circ}$ a -90°.
 - c) Ciclo oscilatorio permanente desde $q_d = 180^{\circ}$, a -180°.
- 7.6.2 En relación con el ejemplo 7.4 posicionar al rotor en los siguientes ángulos deseados para un motor de pasos unipolar bajo el modo de operación 8-step:
 - a) 45° , -45° .
 - b) $90^{\circ}, -90^{\circ}.$
 - c) 135°, -135°.
 - d) 270° , -270°
- 7.6.3 Considere el modo operativo 4-step de un motor a pasos unipolar, posicionar el rotor en los siguientes ángulos deseados:
 - a) 35° , -35° .
 - b) 70° , -70° .
 - c) 115°, -115°.
 - d) 290°, -290°
- 7.6.4 En relación con los cuadros de código Arduino 7.5, 7.5a, 7.5b y 7.5c (ver archivo cap7_movmotor.ino) sobre las subrutinas giro_negativo(int tiempo) y giro_positivo(int tiempo) para manejar el sentido de giro de la rotación de un motor a pasos bajo el modo operativo de 8-step. Unificar las anteriores funciones en una sola rutina que maneje ambos sentidos de giro.

a) Escriba una subrutina que contemple ambas direcciones de giro, por ejemplo de nombre: giro_8step(frecuencia, angulo) donde el argumento de entrada frecuencia define el período de la señal de pulsos para activar las bobinas del motor y el argumento angulo es la referencia deseada (positiva o negativa) expresada en grados.

- 7.6.5 Considere el cuadro de código Arduino 7.6 (archivo cap7_28BYJ4Step.ino) de las subrutinas giro_negativo4step(int tiempo) y giro_positivo4step(int tiempo) que manejan el sentido de giro de la rotación de un motor a pasos bajo el modo operativo de 4-step. Unificar las anteriores funciones para satisfacer los siguientes aspectos:
 - a) Diseñar una subrutina que contemple ambos direcciones de giro, por ejemplo de nombre: giro_4step(frecuencia, angulo) donde el argumento de entrada frecuencia es el periodo de las señal de pulsos para activar las bobinas del motor y el argumento angulo es la referencia deseada (positiva o negativa) para posicionar al rotor expresada en grados.
- 7.6.6 Considere el diagrama electrónico de la figura 7.19; a través de un potenciómetro de 10 K Ω variar el voltaje de la terminal central en forma proporcional al giro del rotor de un motor a pasos. Considere que para el intervalo de 2.5 a 5 V tenga movimientos positivos en el rango de 0° a 360° y para el intervalo de 0 a 2.49 V, el motor gire en el rango de 0° a -360°.
 - a) ¿Cómo establecer el algoritmo para escalar el intervalo de voltaje que sea proporcional al giro del rotor?
 - b) ¿Cómo determinar la velocidad de movimiento del rotor?

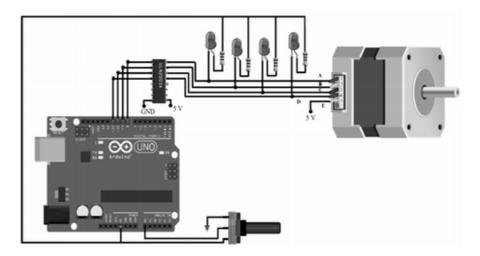
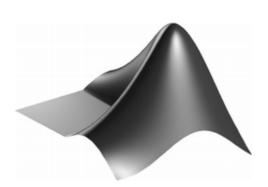


Figura 7.19 Potenciómetro como interface para variar la posición de giro del rotor.



Arduino con Matlab

$\operatorname{Capítulo}$

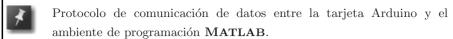


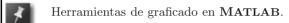
- 8.1 Introducción
- 8.2 Información Arduino en MATLAB
- 8.3 Integración numérica
- 8.4 Diferenciación numérica
- 8.5 Registro de resultados de trabajo
- 8.6 Resumen
- 8.7 Referencias selectas
- 8.8 Problemas propuestos

Competencias

Dominar las herramientas y protocolo de comunicación serial USB para desplegar la información numérica de variables, datos y funciones en el ambiente de programación de MATLAB, mientras se encuentra en ejecución el sketch sobre la tarjeta electrónica Arduino, así como registrar en un archivo de datos esa información para propósitos de análisis, procesamiento y estudio detallado del sistema a controlar o automatizar.

Desarrollar habilidades en:





Registro de información dentro del ambiente de MATLAB.

Registro de información en un archivo de texto en formato tabular.

8.1 Introducción 247

8.1 Introducción



El ambiente de programación Arduino permite la edición, compilación y descarga de código a la tarjeta Arduino para su ejecución; sin embargo, este paquete se encuentra limitado para presentar la información que proviene de la ejecución del sketch en la tarjeta electrónica, es decir de la información que recibe, sólo puede desplegar variables en formato de texto, gráficas, barras o menús no los puede realizar.

La exhibición de datos del sketch se realiza a través de cadenas de texto en la ventana monitor serial \bigcirc , del ambiente de programación Arduino, siendo la única herramienta para presentar datos dentro de dicho ambiente; no presenta gráficas, es decir no cuenta con la posibilidad de observar la evolución en el tiempo de variables o funciones que están manipulando datos por medio de operaciones aritméticas o lógicas en la tarjeta electrónica, tampoco registra o salva la información en archivos de datos.

Las gráficas de variables y datos en función del tiempo en MATLAB no sólo mejoran el aspecto visual de la información, también facilitan la interpretación y análisis del proceso que se está automatizando, detectando aspectos cualitativos que pueden ser corregidos en el sketch para obtener la respuesta deseada de la planta.

Por lo tanto, es muy importante generar un medio de comunicación para intercambio de información entre los ambientes de programación Arduino y MATLAB.

Las herramientas de programación en MATLAB realizan dicho vínculo, ya que cuenta con funciones que permiten construir objetos tipo serial que direccionan el mismo puerto USB que está usando la tarjeta Arduino, por lo que el intercambio de información se realiza a través de mensajes de texto y con la adecuada conversión de datos a tipo char, entero o flotante para el procesamiento en MATLAB, con lo que se obtiene un sistema de control de alto desempeño.

En este capítulo se presenta al usuario el desarrollo de un conjunto de herramientas que le permiten graficar en MATLAB, la información que requiera analizar sobre variables y funciones del sketch que está ejecutándose en la tarjeta Arduino, así como salvar dichos datos en formato tabular en un archivo de texto, conformado por columnas que dependen del número de variables a procesar y con el número de renglones necesarios a almacenar.

Se propone un procedimiento para grabar o registrar resultados de simulación o experimentales en un archivo texto, de esta forma el usuario puede realizar un procesamiento y análisis posterior a la adquisición de datos.



8.2 Información Arduino en Matlab

NA aplicación importante entre el sistema Arduino y MATLAB es la representación gráfica de los datos, variables, funciones que se manipulan computacionalmente cuando se ejecuta un sketch en la tarjeta Arduino. La información de interés para el usuario sobre el algoritmo en ejecución puede ser enviada en forma periódica en el tiempo para su presentación gráfica dentro del ambiente de programación MATLAB; esto se puede llevar a cabo, sin suspender la ejecución del sketch residente en la plataforma electrónica de la tarjeta Arduino. De esta forma, se puede profundizar en el análisis, estudio y comportamiento del sistema o planta que se está controlando.

Diversas aplicaciones se pueden realizar para su representación gráfica, coordinando la ejecución de un sketch en la tarjeta Arduino con un script en MATLAB, tales como:

- Adquisición de datos de sensores.
- Procesamiento de señales.
- Simulación de sistemas dinámicos.
- Estudio cualitativo de algoritmos numéricos.
 - Algoritmos de control: error de posicionamiento, ganancias, derivativas e integrales.

En esta sección se presenta el desarrollo de herramientas computacionales que permitan la adecuada sincronía de comunicación por el puerto serial USB para envío de información de un sketch que se encuentra ejecutando en la tarjeta Arduino y la recepción de esos datos en el ambiente de programación de MATLAB para presentarlos en forma tabular o gráfica. Se describe un sencillo protocolo de información para transmitir serialmente datos de variables y funciones a MATLAB.

♣ ♣ ♣ Ejemplo 8.1

Presentar en forma gráfica en el ambiente de programación **MATLAB** los datos generados por la función $f(t) = \operatorname{sen}(t)$, cuyo código se encuentra ejecutándose en cualquier modelo de tarjetas Arduino. La variable t representa la evolución del tiempo. Para llevar a cabo este planteamiento, se requiere diseñar un sketch que envíe información periódicamente de [y(t), t] a través del puerto USB a un script que se encuentre corriendo en **MATLAB**.

Solución

Dentro de las características principales de los sistemas empotrados (embedded systems) se encuentra

Alfaomega

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

FERNANDO REYES CORTÉS • JAIME CID MONJARAZ

la dedicación exclusiva de ejecutar la tarea o actividad programada, sin tener distractores como desplegado de información en forma tabular o gráfica. Por otro lado, el ambiente de programación Arduino es un paquete de cómputo con herramientas para edición, compilación y descarga del código de máquina del sketch para su ejecución en la tarjeta Arduino; tiene una herramienta modesta para desplegar información de las variables del sketch (monitor serial), este desplegado se realiza por medio de mensajes de texto, no está diseñado para desplegar información en forma gráfica o empleando menús y botones de instrumentación virtual.

Por tal motivo, resulta de particular relevancia el empleo de herramientas gráficas del ambiente de programación **MATLAB** para presentar adecuadamente la información mientras se ejecuta el sketch en la tarjeta Arduino. El envío de información se realiza por medio del puerto USB, se requiere un programa en **MATLAB** que reciba esta información y convierta los datos en formato flotante o enteros para su presentación en forma tabular o gráfica.

Considere el caso donde se tiene la ejecución de la función $y(t) = \operatorname{sen}(t)$ en la tarjeta Arduino a través del sketch **cap8_funciones** cuya descripción se presenta en el cuadro de código Arduino 8.1. El problema consiste en presentar los valores de esa función y(t) en forma gráfica en el ambiente de programación **MATLAB**. La línea 2 contiene la librería para funciones matemáticas, definida en el header o cabecera del sketch. La variable global **y** registra los datos de la función $y(t) = \operatorname{sen}(t)$ y **t** (línea 4) es el argumento de la función y(t); para finalidades de presentación gráfica, el argumento **t** puede tomar valores negativos, cero y positivos.

En la línea 5 se declara la subrutina $envia_datos(\mathbf{x}, \mathbf{y})$ con argumentos del tipo flotante, siendo \mathbf{x} los valores del eje horizontal (eje de las coordenadas), en este caso representa la variable tiempo, \mathbf{y} los valores de la función f(t) en el eje vertical (abscisas). El protocolo de envío de información a través del puerto serie USB es muy simple, pero es conveniente mantenerlo de esta forma para evitar pérdida de datos; consiste en emplear la función $\mathbf{Serial.print}(\mathbf{y})$ para enviar primero un valor de la función y(t), seguido de dos espacios en blanco $\mathbf{Serial.print}("")$, posteriormente el valor del tiempo con retorno de línea $\mathbf{Serial.println}(\mathbf{x})$ (ver línea 8). Es importante destacar que este sencillo protocolo de información permita la correcta sincronización de datos entre la tarjeta Arduino \mathbf{y} la computadora donde radica el ambiente de \mathbf{MATLAB} .

En la línea 10 se encuentra la subrutina de configuración **setup()**, donde se programa la velocidad de transmisión serial (9600 Baudios). El lazo principal del sketch **loop()** inicia a partir de la línea 13.

La función a graficar y = sen(t) se ubica en la línea 14, el envío de información tiempo \mathbf{t} y datos \mathbf{y} se realiza en la línea 24; la evolución del tiempo se realiza por múltiplos de 10 mseg, es decir $\mathbf{t} = \mathbf{t} + 0.01$; observe que se emplea la función $\mathbf{delay}(10)$ para generar el retardo de tiempo adecuado.

El script cap8_graficar descrito en el cuadro de código MATLAB 8.2 se encuentra directamente correlacionado con el sketch cap8_funciones que se ejecuta en la tarjeta Arduino. Las funciones comprendidas entre las líneas 4 a la 7 se utilizan para: liberar espacio de memoria de algunas variables utilizadas por otros programas de MATLAB que aún permanecen residentes (clear all); para cerrar archivos y gráficas que se encuentran abiertos se emplea close all ; la función clc limpia la ventana de comandos y format utiliza desplegado de datos de manera compacta (con 4 fracciones para punto flotante).

En la línea 9 se crea el objeto serial canal_serie del cual el ambiente de programación MATLAB obtendrá los datos para graficar; dentro del protocolo de comunicación con la tarjeta Arduino se encuentra establecer la misma velocidad de transmisión serial que fue definida en el sketch cap8_funciones, para este ejemplo se utiliza 9600 Baudios (ver línea 11 del cuadro de código Arduino 8.1), el envío de datos serial incluye terminación de cadena con retorno de carro y línea de alimentación \r\n ('CR/LF'). Es importante utilizar el mismo canal serial de la tarjeta Arduino, en este ejemplo se presupone que es COM4, esto depende de las características y recursos de la computadora. Dentro del ambiente de programación Arduino, menú de Herramientas, submenú Puerto Serial se puede ubicar el puerto de comunicación serial asociado a la tarjeta Arduino como se indica en la figura 8.1.

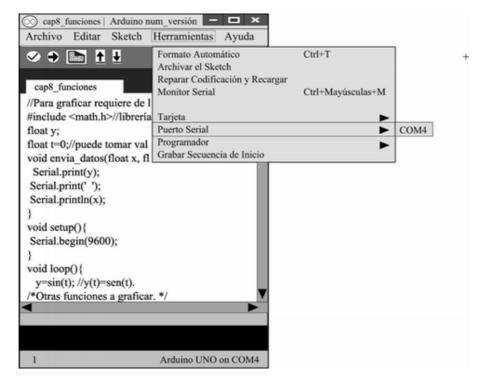


Figura 8.1 Menú de Herramientas del ambiente de programación Arduino.

El canal de comunicación serie se abre con la función **fopen(canal_serie)** como se ilustra en la línea 10; las funciones **grid on** y **hold on** descritas en la línea 12 activan las rejillas y desplegado consecutivo de gráficas, respectivamente; el tipo y forma de líneas gráficas se define en el renglón 13 del script **cap8_graficar**.

La primera adquisición de datos del puerto serie es muy importante, ya que establece la sincronía de comunicación con la tarjeta Arduino, elimina basura almacenada en el puerto serial y se establece un canal seguro de comunicación; se lleva a cabo en la línea 14 por medio de la función **fscanf(...)** donde se leen dos elementos para iniciar protocolo de comunicación serial.

El script cap_graficar presenta en forma gráfica mil puntos de adquisición del sketch cap8_funciones, en la línea 17 el pivote i o contador de número de datos adquiridos se inicializa en 1. A partir de la línea 18 se encuentra el código de adquisición y desplegado gráfico por medio de la instrucción while(i<1000){...}, la adquisición de datos se realiza por medio de la función scanf(...) leyendo del puerto serial los datos de la función y(t) y del tiempo t, almacenando la información en la variable tipo flotante datos, ya que el formato de conversión indicado en la función scanf(...) es del tipo flotante %f. La variable datos es un vector columna con 2 renglones (ver línea 19), es decir:

$$\mathbf{datos} = \begin{bmatrix} \mathrm{datos}(1,\!1) \\ \mathrm{datos}(2,\!1) \end{bmatrix} \in \mathrm{I\!R}^{2 \times 1}$$

En las líneas 20 y 21 se registran los valores de y(t) y t en la variable vector **datos** para incorporarlos en la gráfica por medio de la función **set(...)** como se indica en la línea 22 y su visualización gráfica actualizada en la línea 23 por medio de la función drawnow. El incremento de la variable contador de adquisición i se realiza en la línea 24. La figura 8.2 muestra la representación gráfica de la función $y(t) = \operatorname{sen}(t)$, la cual es ejecutada por el sketch cap8_funciones en la tarjeta Arduino. De las líneas 26 a la 28 se cierra el canal de comunicación serial. Se debe hacer notar al lector que el proceso de graficado no es en tiempo real; sin embargo, se hace en forma instantánea y corresponde a t = 10 segundos de la función y(t).

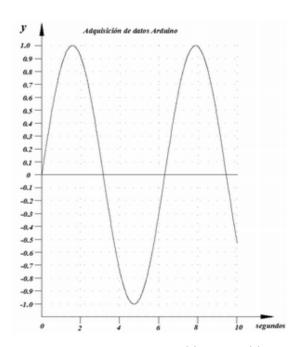


Figura 8.2 Función y(t) = sen(t).

```
    Código Arduino 8.1: sketch cap8_funciones

  Arduino. Aplicaciones en Robótica y Mecatrónica.
  Capítulo 8 Comunicación con MATLAB.
  Fernando Reyes Cortés y Jaime Cid Monjaraz.
  Alfaomega Grupo Editor: "Te acerca al conocimiento".
   Sketch cap8_funciones.ino
 1 //Para graficar requiere de la ejecución del script cap8_graficar desde MATLAB.
 2 #include <math.h>//librería para funciones matemáticas.
 3 float y;
 4 float t=0;//puede tomar valores negativo, 0 y positivos.
 5 void envia_datos(float x, float y){
       Serial.print(y);
 6
       Serial.print(" ");
       Serial.println(x);
 8
 9 }
10 void setup(){
11
       Serial.begin(9600);
12 }
13 void loop(){
       y=\sin(t); //y(t) = \sin(t).
14
       /*Otras funciones a graficar. */
15
       //y=1-pow(2.37,-t); //y(t) = 2.37^{-t}.
16
       //y = t/sqrt(1+t*t); //y(t) = \frac{t}{\sqrt{1+t^2}}.
17
       //y = \tanh(t); //y(t) = \tanh(t).
18
       //y = t/(1 + abs(t)); //y(t) = \frac{t}{1+|t|}.
19
       //y=2*t/(1+t*t); //y(t)=2\frac{t}{1+t^2}.
20
       //y = \sinh(t)/(1 + \cosh(t)) ; //y(t) = \frac{\sinh(t)}{1 + \cosh(t)}
       //y = (1-pow(2.27,-t*t))*t; //y(t) = [1-2.27^{-t^2}]t.
22
23
       //y = t * t * t * t * \sin(t); //y(t) = t^4 \operatorname{sen}(t).
       envia_datos(t,y);//transmisión serial de datos al ambiente MATLAB.
       t=t+0.01; //t=kh, donde h=0.01 y k \in N: k=1,2,\dots,\dots
25
       delay(10); //retardo de 10 mseg (0.01 s).
26
27 }
```

Código MATLAB 8.2 cap8_graficar.m

Arduino. Aplicaciones en Robótica y Mecatrónica.

Capítulo 8 Comunicación con MATLAB.

Fernando Reyes Cortés y Jaime Cid Monjaraz.

Alfaomega Grupo Editor: "Te acerca al conocimiento".

Archivo cap8_graficar.m

Versión de **Matlab** 2014a

- 1 %Este script utiliza el sketch cap8_funciones.
- 2 %Primero ejecutar el sketch cap8_funciones en la tarjeta Arduino.
- 3 %Después ejecutar este script desde al ambiente de programación MATLAB.
- 4 clear all %elimina variables de programas anteriores de memoria.
- 5 close all %cierra gráficas, archivos y objetos abiertos.
- 6 clc %limpia la ventana de comandos de MATLAB.
- 7 format short % despliega datos en formato corto.
- 8 %Crear objeto serie.
- 9 canal_serie = serial('COM4', 'BaudRate', 9600, 'Terminator', 'CR/LF');
- 10 fopen(canal_serie); %abrir puerto serial.
- 11 xlabel ('Segundos'); ylabel ('Datos'); title ('Adquisición de datos Arduino');
- 12 grid on; hold on;
- 13 prop = line(nan,nan, 'Color', 'b', 'LineWidth',1); %tipos de líneas de gráficas.
- 14 datos = fscanf(canal_serie, '%f %f ,[2,1]); %sincronía de datos del puerto serie.
- 15 clc; %limpia ventana de comandos de MATLAB.
- 16 disp('Adquisición de datos de la tarjeta Arduino UNO: utilizar el sketch cap8-funciones');
- 17 i=1; %inicializa pivote de datos a graficar.
- 18 while i<1000 % grafica 1000 puntos de datos de la función f(t) vs t.
- 19 $datos = fscanf(canal_serie, '\%f \%f', [2,1]); \%lee datos del puerto serie.$
- y(i)=datos(1,1);%datos del eje vertical: valores de la función f(t). 20
- x(i)=datos(2,1); %datos del eje horizontal (tiempo). 21
- set(prop, 'YData', y(1:i), 'XData', x(1:i));22
- drawnow; %grafica datos con actualización de información. 23
- i=i+1; %incrementa pivote. 24
- 25 end
- 26 fclose(canal_serie); %cierra objeto serial.
- 27 delete(canal_serie); %libera memoria del objeto serial.
- 28 clear canal_serie; % limpia canal serie.

En el sketch **cap8_funciones** se encuentran otras funciones y(t) que pueden ser graficadas en **MATLAB** y que están deshabilitadas por comentarios de las líneas 16 a la 23. Por ejemplo, las gráficas de las siguientes funciones se presentan en la figura 8.3.

$$y_1(t) = 1 - 2.37^{-t} (8.1)$$

$$y_2(t) = \frac{t}{\sqrt{1+t^2}} \tag{8.2}$$

$$y_3(t) = 2\frac{t}{(1+t^2)} \tag{8.3}$$

$$y_4(t) = \frac{\operatorname{senh}(t)}{1 + \cosh(t)} \tag{8.4}$$

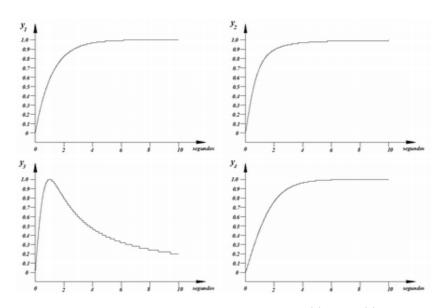


Figura 8.3 Gráficas de funciones $y_1(t) \cdots y_4(t)$.

Los pasos requeridos para ejecutar el sketch **cap8_funciones** en cualquier modelo de las tarjetas Arduino son los que se describen en la sección 2.4 del capítulo 2 **Instalación y puesta a punto del sistema Arduino**. Sin embargo, a continuación se describen en forma resumida:

- Editar el código fuente en lenguaje C del sketch deseado.
- En el menú **Herramientas** del ambiente de programación Arduino seleccionar modelo de tarjeta y velocidad de comunicación serial en 9600 Baudios.
- Compilar el sketch mediante el icono 🕢.
- Descargar el código de máquina a la tarjeta Arduino usando 👈.

Para graficar los datos en el ambiente de programación **MATLAB** tome en cuenta los siguientes aspectos:

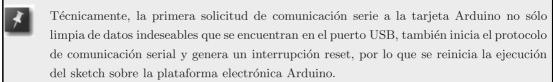


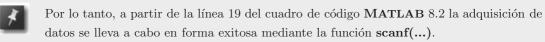
Primero ejecutar el sketch cap8_funciones en la tarjeta Arduino.



Posteriormente, ejecutar el script **cap8_graficar** desde **MATLAB** (oprima el icono **Run** dentro del editor o la tecla F5).

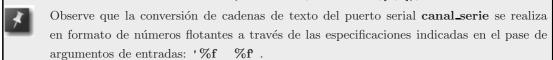






Asimismo, para la adquisición de datos es de suma importancia resaltar la línea 19 del cuadro de código MATLAB 8.2; la función fscanf(...) se utiliza de la siguiente forma:

datos = fscanf(canal_serie, '%f %f', [2,1]);



Lo anterior significa que retorna datos de tipo flotante como un vector de una sola columna y dos renglones.

Esto también se indica en el pase de argumentos del último parámtro de esa misma función: [2,1].

De ahi que el registro de los 1000 valores de la función y(t) y del tiempo t se realiza como: y(i)=datos(1,1), y(i)=datos(2,1) tal y como se indican en las líneas 20 y 21, respectivamente.

8.3 Integración numérica



 \mathbf{E}^{L} método de integración numérica es un procedimiento que permite resolver a través de un algoritmo de cómputo (método numérico Int_k) la solución de una integral $\mathsf{Int}(t)$, la solución que se obtiene no es analítica, más bien es numérica y se puede realizar en lenguaje C y ejecutarse en las tarjetas Arduino.

Considere la siguiente integral definida de una función continua f(t) sobre un intervalo [0, t]:

$$Int(t) = \int_{t_0}^{t} f(\sigma)d\sigma \tag{8.5}$$

donde t es la variable temporal y t_0 es la condición inicial de t.

El método de Euler permite obtener un procedimiento recursivo para resolver la integral (8.5) en forma discreta de la siguiente forma:

$$Int_k = Int_{k-1} + hf(t_k) \tag{8.6}$$

donde Int_k es la integral discreta de Int(t), Int_{k-1} es el valor de la integral discreta en el estado anterior k-1, h es el periodo de muestreo y $f(t_k)$ es el valor de la función f(t) en tiempo discreto $t_k = kh$, siendo $k \in N$ un número natural: $k = 1, 2, 3, \cdots$

♣ ♣ Ejemplo 8.2

Utilizando el método de Euler, desarrollar la solución numérica de la integral:

Int
$$=$$
 $\int_{t_0}^t \sigma^6 d\sigma$.

donde t_0 es la condición inicial de la variable temporal t.



Considere el periodo de muestreo h=0.001, así como el intervalo de integración definido como: $t\in[0,h,\cdots,10].$



Graficar en **MATLAB** la respuesta de la solución recursiva y compararla con la solución analítica de la integral definida $\text{Int}(t) = \frac{1}{7} [t^7 - t_0^7].$

Solución

Sea el sketch cap8_integral definido en el cuadro de código Arduino 8.3, el cual tiene implementado en lenguaje C el algoritmo de integración numérica usando el método de Euler (8.6).

En la línea 1 se utiliza la librería **math.h** para calcular la integral numérica de diversas funciones matemáticas, las líneas comprendidas entre 2 a la 3 se encuentra la declaración de variables globales para la integral analítica, numérica y tiempo discreto, respectivamente. El intervalo de integración se encuentra entre 0 y 10 segundos, el paso de integración es de 1 milisegundo. La variable que define a la función que se va a integrar numéricamente se declara en la línea 6.

La función que establece el protocolo de envío de datos entre la tarjeta Arduino y MATLAB es la subrutina envia_datosI(float x, float y1, float y2) (ver línea 7), el primer argumento x representa la variable tiempo (eje de las coordenadas), mientras que información y1 y y2 contienen los resultados del proceso de integración sobre el eje vertical (eje de las abscisas), respectivamente.

El orden en que se envían los datos los define el usuario; en este ejemplo, primero se envía el tiempo \mathbf{x} , seguido de la integral numérica Int_k , finalmente, la integral definida Int .

Observe que entre cada dato recibido se inserta al menos un espacio en blanco por medio de la función **Serial.print("")**, con la finalidad de indicar al procedimiento en **MATLAB** el término de la cadena de texto de un dato y de esta forma se puedan distinguir los diferentes valores de la variables a registrar en **MATLAB**.

En la función de configuración **setup()** (ver línea 14) se establece la velocidad de comunicación serial en 9600 Baudios y a partir de la línea 17 inicia el lazo principal de programación **loop()** con el algoritmo de integración numérica por el método de Euler.

En primera instancia, se inicializa el intervalo de integración $t_k = 0$ y en la línea 20 se implementa el algoritmo recursivo de integración numérica para el intervalo de integración: $t \in [0, 0.001, 0.002, \dots, 9.99, 10]$:

$$Int_k = Int_{k-1} + ht_k^6 \tag{8.7}$$

Por otro lado, la integral definida de la función $f(t) = t^6$ está dada por

Int
$$=\int_{t_0}^t \sigma^6 d\sigma = \frac{t^7}{7} \bigg|_{t_0}^t = \frac{1}{7} [t^7 - t_0^7]$$
 (8.8)

donde t_0 es la condición inicial del intervalo de integración (para ese caso $t_0=0$).

Sobre la línea 21 se define la función f(t) a integrar numéricamente; como parte de la programación del sketch **cap8_integral** se pueden integrar diversas funciones, de ahí que se requiere la librería de funciones matemáticas **math.h** declarada en la línea 1.

El cálculo de la integral definida Int para la función t^6 se evalúa entre los límites del intervalo

de integración, es decir: Int = $\int_{t_0=0}^{t=10} \sigma^6 d\sigma = \frac{1}{7} [t^7 - t_0^7] = \frac{10^7}{7} = 1.4286e + 06$, este resultado coincide con el proceso recursivo del método de integración numérica de Euler: $\mathrm{Int}_k = \mathrm{Int}_{k-1} + ht^6$ (implementado en la línea 22).

Sin embargo, una forma más exigente para verificar el resultado de integración entre Int y el método de Euler Int $_k$ es comprobar que los cálculos de integración sean iguales para cada punto discreto t_k sobre el intervalo de integración $t_k \in [0, h, \dots, 10]$, por lo que en la línea 23 se programa el cálculo analítico de la integral definida Int de la función $f(t_k) = t_k^6$. El tiempo discreto t_k se incrementa como múltiplos del paso de integración h, es decir: $t_k = t_{k-1} + h$, con h = 0.001 (línea 25).

La figura 8.4 se obtiene en MATLAB y muestra el resultado comparativo entre el algoritmo recursivo de la integral numérica Int_k con el cálculo que proporciona la integral definida Int para cada valor de t_k sobre el intervalo $t \in [0, h, \dots, 10]$ segundos, el paso de integración h es de una milésima de segundo. Observe la similitud del proceso de integración numérica, ambas gráficas Int e Int_k se superponen coincidiendo plenamente en todo el intervalo de integración.

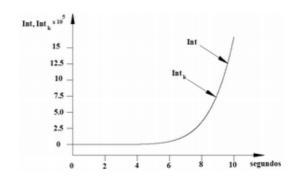


Figura 8.4 Comparación entre Int_k e Int.

Para calcular la integral se debe tomar en cuenta las condiciones iniciales t_0 sobre el intervalo de integración, el cual se ve reflejado en la integral definida $\int_{t_0}^t \sigma^6 d\sigma = \frac{1}{7} [t^7 - t_0^7]$, así como en la integración numérica $\operatorname{Int}_k = \operatorname{Int}_{k-1} + ht_k^6$ en Int_{k-1} para k=1, es decir: $\operatorname{Int}_0 = t_0^7$.

El envío de información a MATLAB se realiza en la línea 24 utilizando el protocolo de información definido en la función envia_datos(...). El sketch cap8_integral se ejecuta en la tarjeta Arduino y envía información de datos para graficar al script cap8_graficaInt descrito en el cuadro de código MATLAB 8.4. Este programa es similar al presentado en el cuadro de código 8.2 (script cap8_graficar).

No obstante, como puede observar en la línea 2 abre el canal sobre el puerto serial COM3 a diferencia del script cap8_graficar que lo hace a través del puerto COM4, esto depende de las características de la computadora donde está conectada la tarjeta Arduino, debido a que el canal serie que abre MATLAB debe ser el mismo puerto serie USB al que está conectada la tarjeta Arduino, es necesario obtener esta información directamente del ambiente de programación Arduino, menú de Herramientas, como se presenta en la figura 8.1.

```
€ Código Arduino 8.3: sketch cap8_integral
   Arduino. Aplicaciones en Robótica y Mecatrónica.
   Capítulo 8 Comunicación con MATLAB.
   Fernando Reyes Cortés y Jaime Cid Monjaraz.
   Alfaomega Grupo Editor: "Te acerca al conocimiento".
   Sketch cap8_integral.ino
 1 #include <math.h>
 2 float Int; //variable utilizada en la integral analítica.
 3 float Int_k=0, t_k=0; //condición inicial: integral numérica Int<sub>k</sub> y variable temporal discreta t<sub>k</sub>.
 4 float h=0.001;//periodo de muestreo.
 5 int k;//número entero natural k \in N.
 6 float funcion;//variable que define el tipo de función.
 7 void envia_datosI(float x, float y1, float y2){
        Serial.print(y1);
        Serial.print(" ");
 9
        Serial.print(y2);
10
11
        Serial.print(" ");
12
        Serial.println(x);
13 }
14 void setup(){
        Serial.begin(9600);
15
16 }
17 void loop(){
        t_k=0;
18
        Int_k=0;//condición inicial de la integral numérica, para k=1, Int<sub>0</sub> = t_0^7.
19
        for (k=0; k<10000; k++){//método iterativo de Euler para integración numérica.
20
           funcion=t_k*t_k*t_k*t_k*t_k*t_k*t_k*/función a integrar: <math>f(t) = t^6.
21
           Int_k=Int_k+h*funcion;//integral por el método de Euler: Int<sub>k</sub> = Int<sub>k-1</sub> + ht_k^6.
22
           Int=t_k*t_k*t_k*t_k*t_k*t_k*t_k/7.0;//integral definida: Int = \int_{t_0}^{t} \sigma^6 d\sigma = \frac{1}{7} [t^7 - t_0^7].
23
\mathbf{24}
           envia_datosI(t_k,Int_k, Int);//envía datos a graficar a MATLAB.
           t_k=t_k+h;//evolución de la variable tiempo discreto.
           delay(h);//retardo por h milisegundos
26
27
28 }
```

Código MATLAB 8.4 cap8_graficaInt.m

Arduino. Aplicaciones en Robótica y Mecatrónica.

Capítulo 8 Comunicación con MATLAB.

Fernando Reyes Cortés y Jaime Cid Monjaraz.

Alfaomega Grupo Editor: "Te acerca al conocimiento".

Archivo cap8_graficaInt.m

Versión de Matlab 2014a

```
1 clear all close all clc
 2 format short
 {\bf 3} \ \ {\rm canal\_serie} = {\rm serial('COM3'} \ \ , {\rm 'BaudRate'} \ \ , 9600, {\rm 'Terminator'} \ \ , {\rm 'CR/LF'} \ \ ); \% crear objeto serie.
 4 fopen(canal_serie); %abrir puerto.
 5 xlabel('Segundos');
 6 ylabel('Datos' );
 7 title ('Adquisición de datos Arduino');
 8 grid on; hold on;
 9 prop = line(nan,nan,'Color', 'b', 'LineWidth',1);
10 datos = fscanf(canal_serie, '%f %f %f ,[3,1]); %leer el puerto serie.
11 clc:
12 disp('Adquisición de datos de la tarjeta Arduino UNO');
13 i=1;
14 while i<10000
15
        datos = fscanf(canal_serie, '%f %f %f',[3,1]); %leer el puerto serie.
        Int_k(i) = datos(1,1);
16
       Int(i) = datos(2,1);
17
        tiempo(i)=datos(3,1);
18
        set(prop, 'YData', Int(1:i), 'XData', tiempo(1:i));
19
20
        drawnow;
21
      i=i+1;
22 end
23 figure
24 plot(tiempo,Int, tiempo, Int_k)
25 fclose(canal_serie); %cierra objeto serial.
26 delete(canal_serie); %libera memoria.
27 clear canal_serie;
```

Otras diferencias entre los scripts **cap8_graficaInt** y **cap8_graficar** se ubican en las líneas 10 y 14 del cuadro de código **MATLAB** 8.4, donde se reciben tres datos y se grafican diez mil puntos, respectivamente; mientas que en el anterior script se realiza para dos datos y se grafican mil puntos (ver líneas 14 y 18 del cuadro 8.2).

Se enfatiza que la línea 10 del cuadro de código **MATLAB** 8.4 tiene la finalidad de iniciar el proceso de comunicación entre **MATLAB** y la tarjeta Arduino, además de limpiar datos residentes en el puerto serial USB.

A partir de la línea 14 se realiza el proceso de adquisición de datos, registrando la información datos(1,1), datos(2,1) y datos(3,1) en la variables Int, Int_k y tiempo que corresponden a los valores numéricos que envía la tarjeta Arduino de la integrales definida, recursiva y la variable temporal t_k , respectivamente.

La adquisición de datos se realiza directamente del canal serie por medio de la función $\mathbf{fscanf}(...)$ en la línea 15, convirtiendo las tres cadenas de texto en datos de tipo flotante por medio de los comandos '%f %f %f .

El resultado se registra en la variable **datos** el cual es un vector de una columna y tres renglones. La representación gráfica se realiza actualizando datos dentro del intervalo de integración para Int_k vs t_k , por medio de la función **drawnow** (línea 20), un total de diez mil punto son graficados para el intervalo de integración $t \in [0, 0.001, \dots, 10]$.

En la línea 24 se muestra la gráfica comparativa entre los resultados Int e Int_k como se indica en la figura 8.4. Finalmente, a partir de la línea 25 se cierra, borra y limpia el canal serial.

Para graficar los datos en el ambiente de programación **MATLAB** llevar a cabo los siguientes pasos:



Primero ejecutar el sketch **cap8_integral** en la tarjeta Arduino; ver el procedimiento descrito en la página 254.



Posteriormente, ejecutar el script **cap8_graficaInt** desde **MATLAB** (oprimir el icono **Run** dentro del modo de edición o la tecla F5).



8.4 Diferenciación numérica

L' método de diferenciación numérica es una herramienta algorítmica que permite calcular la derivada $\dot{f}(t) = \frac{d}{dt}f(t)$ de una función continua f(t) a través de métodos numéricos, donde t es la variable tiempo.

Hay varios métodos numéricos para obtener la derivada de una función f(t), entre ellos diferenciación numérica es el método de Euler, resulta ser muy utilizado, por su sencillez y eficacia. Este método consiste en realizar la siguiente ecuación:

$$\dot{f}(t) \simeq \dot{f}_k = \frac{f(t_k) - f(t_{k-1})}{h}$$
 (8.9)

donde \dot{f}_k es la aproximación de la derivada de la función f(t), la variable temporal es t, representando el tiempo continuo, t_k es el tiempo discreto, el cual evoluciona como múltiplos del periodo de muestreo h, es decir: $t_k = t_{k-1} + h$.



Método de Euler

En robótica y en sistemas mecatrónicos es frecuente el uso del método de Euler o de diferenciación numérica, ya que por lo general los robots manipuladores, robots móviles, etc., no incorporan sensor de velocidad, sólo el encoder, proporcionando información del desplazamiento articular \boldsymbol{q} , por lo que la estimación de la velocidad $\dot{\boldsymbol{q}}$ del robot se realiza por medio del siguiente algoritmo:

$$\dot{\boldsymbol{q}}_k = \frac{\boldsymbol{q}_k - \boldsymbol{q}_{k-1}}{h}$$

donde h representa el periodo de muestreo del sistema electrónico digital de control.

Observe de la ecuación (8.9), el proceso inverso será la integración de la velocidad o derivada de la función f(t), el cual se obtiene despejando $f(t_k)$ de la siguiente forma:

$$f(t_k) = f(t_{k-1}) + h\dot{f}_k$$
 (8.10)

esta formulación corresponde a la ecuación (8.6), es decir, al método numérico de Euler que se emplea en integración numérica. Las ecuaciones (8.6) y (8.10) son equivalentes entre sí y se utilizan dependiendo si el caso requiere integración o diferenciación numérica, respectivamente. Arduino es una plataforma electrónica viable para implementar esos esquemas.

♣ ♣ Ejemplo 8.3

Implementar el método de diferenciación numérica (método de Euler) para obtener la derivada $\frac{d}{dt}f(t)$ de la función $f(t)=1-e^{-t}$, donde t es la variable de tiempo definida sobre el intervalo $t \in [0, h, 2h, \cdots, 10]$. Considere el desarrollo del método de Euler en un sketch que se ejecute sobre la tarjeta Arduino UNO y los resultados se registren en forma gráfica en el ambiente de programación **MATLAB**.

Solución

La programación en lenguaje C del método de diferenciación numérica se encuentra desarrollado en el sketch **cap8_derivada**, cuya descripción se presenta en el cuadro de código Arduino 8.5.

La función a derivar está dada por: $f(t) = 1 - e^{-t}$ y la derivada $\dot{f}(t) = \frac{d}{dt}[1 - e^{-t}] = e^{-t}$, donde $t \in [0, h, 2h, \dots, 10]$.

La derivada $\dot{f}(t)$ se aproximará de la siguiente forma:

$$\dot{f}(t) \simeq \operatorname{derivada} = \frac{f(t_k) - f(t_{k-1})}{t_k - t_{k-1}}$$
 (8.11)

donde la variable **derivada** representa la aproximación discreta de la derivada continua $\dot{f}(t)$, esta aproximación se realiza por el método de Euler; t_k es la evolución del tiempo en forma discreto y su incremento se lleva a cabo como múltiplos del periodo de muestreo h, es decir: $t_k = kh$, donde k es un número natural $(k \in N)$; en forma recursiva: $t_k = t_{k-1} + h$, como se ilustra en la línea 24.

Observe que la longitud de intervalo de tiempo entre dos muestras consecutivas $f(t_k)$ y $f(t_{k-1})$ es $h = t_k - t_{k-1}$, siendo $f(t_k)$ el valor de la función evaluada en el k-ésimo tiempo discreto, mientras que $f(t_{k-1})$ se evalúa en la muestra k-ésima anterior.

El período de muestreo h que emplea el método de Euler se define de un milisegundo (ver línea 3); el algoritmo de estimación de la velocidad por diferenciación numérica se encuentra implementado a partir de la línea 19.

La variable funcion registra los datos de $f(t) = 1 - e^{-t_k}$ (línea 20) y la derivada se calcula en la línea 21; después de este cálculo se requiere conservar el valor actual, ya que en la siguiente iteración se convierte en la muestra anterior $f(t_{k-1})$, por lo tanto se realiza la siguiente asignación: funcion_ka=funcion_k (ver línea 22).

```
○ Código Arduino 8.5: sketch cap8_derivada

  Arduino. Aplicaciones en Robótica y Mecatrónica.
  Capítulo 8 Comunicación con MATLAB.
  Fernando Reyes Cortés y Jaime Cid Monjaraz.
  Alfaomega Grupo Editor: "Te acerca al conocimiento".
   Sketch cap8_derivada.ino
 1 //Para graficar en MATLAB usar el programa cap8_graficaInt.m
 2 #include <math.h>
 3 float t_k=0, h=0.001; //variable temporal discreta y periodo de muestreo.
 4 int k;//número entero natural.
 5 float derivada, funcion_k, funcion_ka;//función a derivar.
 6 void envia_datosI(float x, float y1, float y2){
       Serial.print(y1);
       Serial.print(" ");
 8
       Serial.print(y2);
10
       Serial.print("")
11
       Serial.println(x);
12 }
13 void setup(){
14
       Serial.begin(9600);
15 }
16 void loop(){
17
       t_{-}k=0;
       funcion_ka=0;
18
       for (k=0; k<10000; k++)//diez mil iteraciones del método de Euler.{
19
          funcion_k=1-exp(-t_k);// función a derivar: f(t_k) = 1 - e^{-t_k}
20
          derivada=(funcion_k-funcion_ka)/h;//diferenciación numérica: \frac{f(t_k)-f(t_{k-1})}{h}.
21
          funcion_ka=funcion_k;//f(t_{k-1}) = f(t_k)
          envia_datosI(t_k,funcion_k, derivada);//envía datos para graficar en MATLAB.
23
          t_k=t_k+h;//evolución de la variable tiempo: t_k=t_{k-1}+h.
24
          delay(h);//en la línea 24, en lado derecho del signo igual \mathbf{t}_{-\mathbf{k}} representa t_{k-1}.
25
26
27 }
```

Es importante aclarar que en la línea 24 $\mathbf{t}_{-\mathbf{k}}$ + \mathbf{h} , antes de asignar el resultado a $\mathbf{t}_{-\mathbf{k}}$ (en el lazo izquierdo del signo igual), la adición aritmética entre la variable $\mathbf{t}_{-\mathbf{k}}$ y el periodo de muestreo \mathbf{h} (en el lado derecho del signo igual), el término $\mathbf{t}_{-\mathbf{k}}$ representa la muestra k-ésima anterior: t_{k-1} , por lo que, para este caso, no se requiere una variable adicional, diferente al escenario que presentan funcion y funcion_ \mathbf{k} a.

Para graficar en MATLAB los resultados obtenidos del sketch cap8_derivada se debe utilizar el script cap8_graficaInt (ver cuadro de código MATLAB 8.4 del ejemplo 8.2). El protocolo de envío de información de la tarjeta Arduino hacia MATLAB se realiza por medio de la subrutina envia_datosI(float x, float y1, float y2) como se detalla en la línea 6 del cuadro de código Arduino 8.5, donde x representa el tiempo discreto t_k , y1 es la función $f(t_k) = 1 - e^{-t_k}$ y la derivada de $f(t_k)$ en y2. El orden en que se envían los datos son: primero la función f(t), posteriormente la derivada numérica y al último el tiempo t_k , de tal forma que la correspondencia al script cap8_graficaInt la variable Int_k corresponde a f(t), Int denota la derivada de f(t) y tiempo a t_k como está indicado en las líneas 16, 17 y 18, del cuadro 8.4, respectivamente. El envío de información no es en tiempo real.

Los resultados generados del método de Euler se presentan en la figura 8.5 (gráfica para $f(t_k)$ y $\dot{f}(t_k)$ usando diez mil puntos):

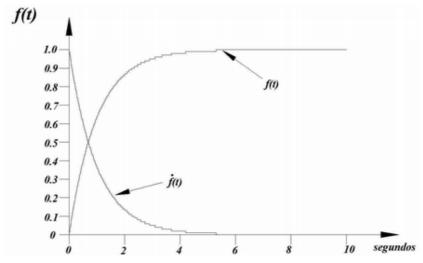


Figura 8.5 Gráficas de $f(t_k) = 1 - e^{-t_k}$ y $\dot{f}(t_k) = e^{-t_k}$.

Consulte la página 254 para ejecutar el sketch cap8_derivada y graficar en MATLAB.



8.5 Registro de resultados de trabajo

Resultados de simulaciones y de experimentos obtenidos a través de robots manipuladores y sistemas mecatrónicos requieren ser registrados en un formato simple para su representación gráfica, interpretación, análisis y estudio.

La plataforma de las tarjetas electrónicas Arduino tiene limitaciones de memoria para almacenar grandes cantidades de datos, en primera instancia, se encuentran dedicados al proceso; esta característica es un rasgo distintivo de los sistemas empotrados. Por tal motivo, es importante contar con herramientas de programación que faciliten el proceso de almacenamiento o registro de la información, sin distraer la atención del algoritmo de control, modelo o esquema en ejecución.

Dentro de las ventajas que tienen las tarjetas Arduino, se encuentra la interfaz de comunicación serial a través del puerto USB; hoy en día, este puerto es una forma común y eficiente para enlazar sistemas digitales. Dentro de los posibles formatos para registrar datos de interés del proceso o planta de estudio se propone el ejemplo de la tabla 8.1; consta de un archivo de texto (ASCII) con varias columnas que representan diversas variables y un número determinado de renglones con la información de las respectivas variables.

Tabla 8.1 Registro de información para simulación/experimentos.

| Número de datos i | Tiempo t | Función $f(t)$ | Derivada $\dot{f}(t)$ | Integral $\int_{t_0}^t \sigma d\sigma$ |
|---------------------|------------|----------------|-----------------------|--|
| 1 | 0.0000 | 0.0000 | 1.0000 | 0.0000 |
| 2 | 0.0025 | 0.0025 | 0.75600 | 0.8000 |
| 3 | 0.0050 | 0.0050 | 1.1000 | 0.9000 |
| 3 | 0.0075 | 0.0036 | 0.8900 | 1.3000 |
| 4 | 0.0100 | 0.0034 | 1.2300 | 2.6000 |
| 5 | 0.0125 | 0.0167 | 1.4300 | 3.1100 |
| : | ÷ | : | : | : |
| 10000 | 10.0000 | 33.8291 | 9.3840 | 50.1630 |

Este formato representa un estilo tabular sencillo y útil para registrar datos numéricos de simulación y experimentos; en esta sección se desarrollan las herramientas para almacenar en archivo de texto datos usando MATLAB a través del puerto USB, mientras el sketch se ejecuta en la tarjeta Arduino.

♣ ♣ ♣ Ejemplo 8.4

Ejecutar en la tarjeta electrónica Arduino UNO, el método de Euler para obtener la derivada e integral de la función rampa f(t)=t sobre el intervalo de tiempo $t\in[0,h,2h,\cdots,10]$ segundos:



Enviar la información de la función f(t), $\dot{f}(t)$ y la $\int_{t_0}^t f(\sigma)d\sigma$ al ambiente de programación **MATLAB** para su representación gráfica.



Registrar en un archivo de datos numéricos con formato tabular como el propuesto en la tabla 8.1.

Solución

En este ejemplo se propone una herramienta para registrar datos numéricos que provienen de simulaciones o experimentos, facilitando no sólo el resguardo de la información, también el proceso de análisis, estudio e interpretación de datos posterior a la adquisición de datos.

Para realizar este ejemplo, se proporciona un procedimiento en **MATLAB** para cargar a memoria la información contenida en el archivo de registro.

El sketch **cap8_Registro** del cuadro de código Arduino 8.6 describe cómo sincronizar el envío de información entre la tarjeta Arduino y las herramientas de graficado y registro de datos en **MATLAB** usando el script **cap8_registrar** (ver cuadro 8.7).

El procedimiento para que ambos programas trabajen de manera coordinada se describe a continuación:



Primero, iniciar la ejecución del sketch cap8_Registro en la tarjeta Arduino.

Desde el ambiente de programación Arduino realizar la compilación y descarga del sketch.

Ver resumen de procedimiento en la página 254.



Segundo, ejecutar desde el ambiente de programación MATLAB el script cap8_registrar.



Tomar en cuenta que la velocidad de transmisión por comunicación serie debe ser la misma por ambos programas, así como el mismo puerto USB que está usando la tarjeta Arduino, deberá estar asignado al programa de MATLAB.

Si el programa en MATLAB se ejecuta antes que el sketch tome en cuenta lo siguiente:



Si el sketch se descarga a través del ambiente de programación Arduino, entonces indicará que el puerto serial no está disponible, debido a que se encuentra ocupado por otra aplicación, en este caso por MATLAB.



Si el sketch ya fue descargado previamente, es decir se encuentra residente en la tarjeta Arduino, entonces únicamente energizar la tarjeta y el enlace de comunicación con el script en MATLAB cap8_registrar se realizará en forma automática.

En el caso de que la sincronía entre el programa en **MATLAB** y el sketch que ya está ejecutándose en la tarjeta Arduino demore más de 10 segundos, en la ventana de comandos de **MATLAB** aparecerá el siguiente mensaje:



Warning: Unsuccessful read: A timeout occurred before the Terminator was reached.



Para solucionar este problema, oprima el botón de reset de la tarjeta Arduino por un segundo y la comunicación se restablecerá.

En la línea 24 del cuadro 8.6 se establece un semáforo de control para coordinar el enlace de comunicación estre los programas en MATLAB y el que se encuentra ejecutándose sobre la tarjeta Arduino. En la fase inicial se emplea bandera igual a uno, enviando el comando 33 al programa de MATLAB para indicarle fase de arranque en la comunicación (ver cuadro 8.7, línea 10). Posteriormente el sketch cap8_Registro esperará que el programa en MATLAB responda con el comando 95 (se envía en la línea 15, cuadro 8.7); en la tarjeta Arduino se detecta este comando a través de la función Serial.available(), la cual sensa si existen datos disponibles del puerto USB, como se ilustra en la línea 27. Cuando la condición fase==95 es verdadera, se limpia el semáforo bandera=0, para no repetir esta sección de código por la función loop(); observe que en la línea 34 del cuadro 8.6 se envía el número de datos al programa en MATLAB para graficar y grabar la información en archivo. Después de un retardo de 0.6 segundos, se ejecutan los algoritmos de integración y diferenciación numérica.

Los diagramas de flujo para la fase de sincronización entre los programas cap8_registrar y cap8_Registro se presentan en las figuras 8.6a y 8.6b, respectivamente.

En la línea 37 del cuadro 8.6 se inicializan en cero a las variables que emplean los algoritmos de integración y diferenciación numérica. El procesamiento de estos algoritmos se realiza a partir de la línea 40 por un número igual de iteraciones definido por num_datos (en este ejemplo, diez mil veces). El envío de información al programa MATLAB se realiza a través de la función envia_datosII(k,t_k,funcion_k,Int_k,Derivada_k) (línea 45), cuyo código se encuentra descrito

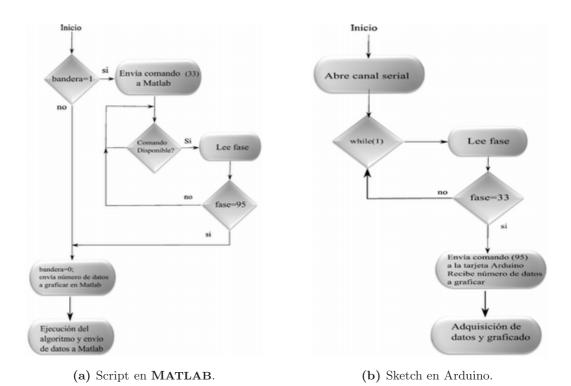


Figura 8.6 Secuencia de la fase de sincronización.

en la línea 9. La evolución del tiempo discreto t_k se realiza como múltiplos del periodo de muestreo h, como se indica en la línea 46.



Protocolo de comunicación

El protocolo de comunicación es el conjunto de reglas que permite enlzar la tarjeta Arduino con el ambiente de programación MATLAB; en el sitio Web de esta obra se describe el protocolo de comunicación que permite el intercambio de datos con MATLAB y la ejecución del sketch en la tarjeta Arduino.

```
€ Código Arduino 8.6: sketch cap8_Registro
  Arduino. Aplicaciones en Robótica y Mecatrónica.
  Capítulo 8 Comunicación con MATLAB.
  Fernando Reyes Cortés y Jaime Cid Monjaraz.
   Alfaomega Grupo Editor: "Te acerca al conocimiento".
   Sketch cap8_Registro.ino
 1 //Para graficar en MATLAB usar el programa cap8_registrar.m
 2 #include <math.h>
 3 float t_k=0,Int_k=0; //condición inicial del tiempo discreto y de la integral numérica.
 4 float h=0.001;//paso de integración.
 5 int k, fase=0, bandera=1;//fase del proceso y bandera de semáforo.
 6 float funcion_k, funcion_ka=0;//función a derivar y su registro anterior.
 7 float Derivada_k;//para registrar la derivada de la función.
 8 int num_datos=10000;//número de datos a graficar en MATLAB.
 9 void envia_datosII(int k, float tiempo, float y1, float y2, float y3){
       Serial.print(k);//número de puntos k.
10
11
       Serial.print(" ");//espacio en blanco, para separar datos de la cadena de caracteres.
12
       Serial.print(tiempo);//variable tiempo t_k
13
       Serial.print(" ");//espacio en blanco, para separar datos de la cadena de caracteres.
       Serial.print(y1);//función f(t_k)
14
       Serial.print(" ");//espacio en blanco, para separar datos de la cadena del mensaje.
15
       Serial.print(y2);// integral numérica de f(t_k).
16
       Serial.print(" ");//espacio en blanco, para separar datos de la cadena del mensaje.
17
18
       Serial.println(y3);// derivada numérica de f(t_k).
19 }
20 void setup(){
21
       Serial.begin(9600);
22 }
23 void loop(){
       if (bandera){
24
          Serial.println(33,DEC);
25
          do{
26
             if(Serial.available()>0){
27
                fase=Serial.read();
```

Continúa código Arduino 8.6a: sketch cap8_registro

Arduino. Aplicaciones en Robótica y Mecatrónica.

Capítulo 8 Comunicación con MATLAB.

Fernando Reyes Cortés y Jaime Cid Monjaraz.

Alfaomega Grupo Editor: "Te acerca al conocimiento".

Continuación del sketch cap8_registro.ino

```
29
                if (fase==95)
                    break;
30
             }
31
          \}while(1);
32
          bandera=0;
33
          Serial.println(num_datos);
34
          delay(600);
35
36
       t_k = 0;
37
38
       funcion_ka=0;
39
       Int_k=0;
       for (k=0; k<\text{num\_datos}; k++){
40
          funcion_k=t_k;//función a integrar y derivar.
41
          Int_k=Int_k+h*funcion_k;//integral numérica por el método de Euler.
42
          Derivada_k= (funcion_k-funcion_ka)/h;//derivada por diferenciación numérica.
43
          funcion_ka=funcion_k;//registro del valor anterior de la función.
44
          envia_datosII(k,t_k,funcion_k, Int_k, Derivada_k);// graficar en MATLAB.
45
          t_k=t_k+h;//evolución de la variable tiempo t_k.
46
47
          delay(h);
48
       }
49 }
```



La compilación y descarga del código de máquina a la tarjeta Arduino del sketch cap8_registro se explica en la página 254.



El cuadro de código **MATLAB** 8.7 contiene el script **cap8_registrar** con las herramientas de graficado y registro de la información en archivo de texto.

```
Código MATLAB 8.7 cap8_registrar.m
  Arduino. Aplicaciones en Robótica y Mecatrónica.
  Capítulo 8 Comunicación con MATLAB.
  Fernando Reyes Cortés y Jaime Cid Monjaraz.
  Alfaomega Grupo Editor: "Te acerca al conocimiento".
   Archivo cap8_registrar.m
                                                          Versión de Matlab 2014a
 1 %Con arduino emplear el programa cap8_registro.ino
 2 clear all close all clc format short
 3 canal-serie = serial('COM3', 'BaudRate', 9600, 'Terminator', 'CR/LF'); %crear objeto serie.
 4 fopen(canal_serie); %abrir puerto serial USB.
 5 xlabel ('Segundos'); ylabel ('Datos'); title ('Adquisición de datos Arduino');
 6 grid on; hold on; fase=0; %inicia sincornía con la tarjeta Arduino.
 7 while (1)
       disp('Sincronizando enlace con tarjeta Arduino UNO')
 8
       [fase, contador]=fscanf(canal_serie, '%d',[1,1]); %lee el puerto serial.
       if (fase==33)
10
           disp('Inicia adquisición de datos con la tarjeta Arduino UNO')
12
           break;
       end
13
14 end
15 fwrite(canal_serie, 95, 'int');
16 num_datos=fscanf(canal_serie, '%d',[1,1]);%número de datos a graficar y registrar.
17 prop = line(nan,nan, 'Color', 'b', 'LineWidth',1);
18 datos = fscanf(canal_serie, '%f %f %f ,[3,1]); %leer puerto serial USB.
19 disp(['Número de datos a graficar: ', num2str(num_datos)]);
20 for i=1:(num_datos-1)
       datos = fscanf(canal_serie, '%i %f %f %f %f ,[5,1]); %leer puerto serie.
21
       num\_puntos(i,1)=datos(1,1); tiempo(i,1)=datos(2,1);
22
       funcion(i,1) = datos(3,1); integral(i,1) = datos(4,1);
23
       derivada(i,1) = datos(5,1);
25
       set(prop, 'YData', integral(1:i,1), 'XData', tiempo(1:i,1));
       drawnow;
26
```

27 end



Continúa código MATLAB 8.7a: cap8_registrar.m

Arduino. Aplicaciones en Robótica y Mecatrónica.

Capítulo 8 Comunicación con MATLAB.

Fernando Reyes Cortés y Jaime Cid Monjaraz.

Alfaomega Grupo Editor: "Te acerca al conocimiento".

Continuación de cap8_registrar.m

- 28 figure
- 29 plot(tiempo, funcion, tiempo, derivada, tiempo, integral)
- 30 fclose(canal_serie); %cierra objeto serial.
- 31 delete(canal_serie); %libera memoria. clear canal_serie;
- 32 resultados=[num_puntos, tiempo, funcion, derivada, integral];
- 33 archivo = fopen('c:\Arduino\Sketchs\archivo_de_datos.dat', 'wt');
- **34** fprintf(archivo, '%i %3.4f %3.4f %3.4f %3.4f \n', resultados');
- **35** fclose(archivo);
- 36 %save —ascii 'c:\Arduino\Sketchs\archivo_de_datos.dat' resultados
- 37 clear resultados;
- 38 type('c:\Arduino\Sketchs\archivo_de_datos.dat')

La adquisición de datos se realiza a partir de la línea 20 del cuadro 8.7, programa cap8_registrar; la información que se registra es la cantidad de puntos enviados (num_puntos(i,1)), tiempo(i,1), los valores de la funcion(i,1), integral(i,1) y derivada(i,1).

Una vez que se registran los num_datos, se prepara la información a grabar en archivo en: resultados=[num-puntos, tiempo, funcion, derivada, integral] como se presenta en la línea 32. En este ejemplo, el nombre del archivo de datos es: archivo_de_datos.dat, es de tipo texto y se genera con la instrucción fopen(...) como se presenta en la línea 33, la trayectoria dentro del disco duro es opcional y la define el usuario.



El formato tabular del archivo de datos se realiza de acuerdo con el que sugiere en la tabla 8.1, esto se consigue con la función **fprintf(...)**.



La primera columna indica el número de renglón de cada dato en formato entero, seguido de 4 columnas en punto flotante con 4 fracciones, por medio de: ' %
i %3.4f %3.4f %3.4f \n' se obtiene la tabulación deseada tal y como se indica en la línea 34.



Note que la variable **resultados** debe ir transpuesta (**resultados**'), para generar adecuadamente los renglones de datos.

Una vez que se han registrado los datos en el archivo de texto, es conveniente cerrarlo para poderlo utilizar (línea 35). También se recomienda liberar la memoria asignada a la variable **resultados** como se ilustra en la línea 37. Una forma inmediata de visualizar la información del archivo de datos es por medio de la función **type(...)**.

Un código opcional para obtener el mismo formato tabular (tabla 8.1) en el archivo de datos, es el que está entre comentarios sobre la línea 36, utilizando la función save. Observe que para este caso, la variable resultados no debe ir transpuesta. Sin embargo, la estética de registro de datos sobre archivo_de_datos.dat no es tan elegante como con el que se obtiene empleando fprintf(...).

En el cuadro de código MATLAB 8.8 se encuentra un sencillo procedimiento denominado cap8_registrarI que muestra la forma de descargar en memoria de la computadora la información registrada en archivo_de_datos.dat y de esta manera, analiza, grafica e interpreta la información correspondiente al proceso de simulación o evaluación experimental con robots o sistemas mecatrónicos.

En la figura 8.7 se muestra la evolución en el tiempo de las variables del sketch **cap8_registro**, el cual ejecuta sobre la tarjeta Arduino los algoritmos de diferenciación e integración numérica por el método de Euler.



Adquisición de datos MATLAB

En el sitio Web de este libro, se encuentran varios programas en lenguaje C que realizan adquisición de datos de sensores de temperatura, celdas solares, encoders, etc., cuya información se transmite serialmente a MATLAB para su presentación gráfica y registro tabular en archivo de datos .



Arduino desde Matlab

En el sitio Web de la presente obra, puede consultar ejemplos de programación y descarga de código Arduino desde el ambiente de programación **MATLAB**.

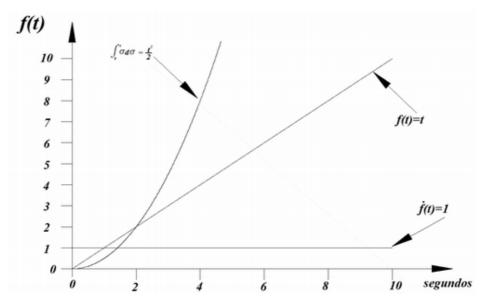


Figura 8.7 Gráficas de f(t) = t, $\dot{f}(t) = \frac{d}{dt}f(t) = 1$ y la $\int_0^t \sigma d\sigma = \frac{t^2}{2}$.



A Código MATLAB 8.8 cap8_registrarI.m

Arduino. Aplicaciones en Robótica y Mecatrónica.

Capítulo 8 Comunicación con MATLAB.

Fernando Reyes Cortés y Jaime Cid Monjaraz.

Alfaomega Grupo Editor: "Te acerca al conocimiento".

Archivo cap8_registrarI.m

Versión de Matlab 2014a

- 1 clc clear all close all
- 2 format short
- **3** datos=load('c:\fer\Arduino\Sketchs\archivo_de_datos.dat');
- 4 tiempo=datos(:,2);
- 5 funcion=datos(:,3);
- 6 derivada=datos(:,4);
- 7 integral=datos(:,5);
- 8 [n, m]=size(tiempo);
- 9 disp(['Número de datos registrados en archivo:=', num2str(n)])
- 10 plot(tiempo, funcion, tiempo, derivada, tiempo, integral)



8.6 Resumen

🖪 N el presente capítulo se han desarrollado procedimientos para enviar datos desde el sketch que I se encuentra ejecutándose en la tarjeta Arduino hacia el ambiente de programación ${f MATLAB}$ para su representación gráfica y registro de la información en archivo con formato tabular.

Para graficar datos en MATLAB se requieren dos archivos: uno de ellos es el sketch que se ejecuta en la tarjeta Arduino, donde se destaca la sección del código correspondiente al protocolo de envío de información a MATLAB; como ejemplo, considere el código 8.1, el cual presenta el formato para enviar 4 variables: tiempo t_k y tres variables más y_1, y_2, y_3 . Observe que se emplean las funciones Serial.print para enviar las variables específicas, después de la transmisión de alguna variable se inserta un espacio en blanco Serial.print("") para separar los datos entre sí, dentro de la cadenas de caracteres del mensaje de texto que conforman el grupo de variables, el último dato a enviar se inserta nueva línea con retorno de carro para indicar la finalización de la cadena del mensaje usando la función **Serial.println(...)**:

€ Código ejemplo 8.1

Protocolo para enviar información desde Arduino a MATLAB.

```
void envio_datos_serial(float tiempo, float y1, float y2, float y3){
    Serial.print(tiempo);//se transmite valores de la variable tiempo t_k
    Serial.print(" " );//un espacio en blanco, para separar los datos.
    Serial.print(y1);//se transmite valores de la función: f_1(t_k)
    Serial.print(" " );//un espacio en blanco, para separar los datos.
    Serial.print(y2);//se transmite valores de la función: f_2(t_k)
    Serial.print(" " );//un espacio en blanco, para separar los datos.
    //La última variable a transmitir debe incluir la función Serial.println(...).
    Serial.println(v3); //f_3(t_k) se transmite con nueva línea y retorno de carro.
```

El archivo que complementa la comunicación se encuentra en MATLAB; se hace particular énfasis en la parte del código que recibe los datos por medio de la función fscanf(...), empleando los siguientes argumentos o parámetros de entrada:



```
datos = fscanf(canal_serie, '%i %f %f %f ,[4,1]);
```

donde $datos \in \mathbb{R}^{4 \times 1}$ es un vector de cuatro renglones y una columna. El número de variables que se envían desde la tarjeta Arduino debe ser el mismo número de renglones de datos.

Recuerde que el aspecto clave es utilizar la función **fscanf(...)** en dos fases, la primera es antes de la adquisición de datos, teniendo la finalidad de inicializar el enlace de comunicación USB y al mismo tiempo limpiar posibles datos parásitos dentro del puerto serial.

La segunda fase se ubica dentro del ciclo de instrucciones que realiza la adquisición de datos para graficado y registro de la información.

8.7 Referencias selectas



A bibliografía para MATLAB es muy amplia, sin embargo, para complementar el material presentado en este capítulo se recomienda al lector revisar las siguientes referencias:

El lenguaje de programación de MATLAB se puede consultar en:



The MathWorks. "Matchmatics: The language of technical computing". The MathWorks. 2005.



http://www.mathworks.com

Desarrollo de métodos numéricos en lenguajes C y C++ pueden obtenerse en:



William H Press, Saul A. Teukoisky, William T. Vetterling, & Brian P. Flannery. "Numerical recipes in C++". Cambridge University Press. 2002.



William H Press, Saul A. Teukoisky, William T. Vetterling, & Brian P. Flannery. "Numerical recipes in C". Cambridge University Press. 2002.

Lenguaje MATLAB y sus aplicaciones en robótica y mecatrónica se describen en:



Fernando Reyes Cortés. "MATLAB Aplicado a Robótica y Mecatrónica". Alfaomega Grupo Editor. 2012.

8.8 Problemas propuestos



M ÉTODOS numéricos es una herramienta importante en robótica, mecatrónica y en general en la automatización de sistemas. Particularmente, Arduino como sistema empotrado tiene el potencial de programar técnicas numéricas en lenguaje C.

A continuación se presenta un conjunto de problemas con la finalidad de ponderar los conocimientos adquiridos por el lector en este capítulo.

8.8.1 El cálculo de la integral $\int_{t_0}^t f(\sigma)d\sigma = \text{Int}(t) - \text{Int}(t_0)$ depende de las condiciones iniciales del intervalo de integración t_0 y del valor de la integral de la función f(t) en dicha condición inicial.

Considere el ejemplo 8.2, compruebe los resultados entre la integración numérica Int_k y la integral definida Int de la función t^6 se satisfacen para las siguientes condiciones iniciales:

- a) $t_0 = 0.001$.
- b) $t_0 = 0.03$.
- c) $t_0 = 0.912$.
- d) $t_0 = 9.81$.

Tome en cuenta el intervalo de integración $t \in [t_0, h, 2h, \cdots, 10]$, con h=0.001.

- 8.8.2 Obtener la derivada numérica (usando el método de Euler) de las siguientes funciones $f(t_k)$:
 - a) $\cos(w t_k)$.
 - b) $\operatorname{sen}(w \ t_k)$.
 - c) $\tanh(w t_k)$.
 - d) $senh(w t_k)$.
 - e) $\cosh(w t_k)$.
 - f) atan $(w t_k)$.
 - g) $\cot (w t_k)$.
 - h) $tanh (sen(w t_k)).$
 - i) $e^{-(w t_k)}$.
 - j) $e^{\operatorname{sen}(w\ t_k)}$.

donde $w \in \mathbb{R}_+$, sea $w = \pi$.

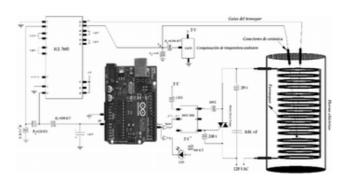
Compare el resultado numérico con las correspondientes derivadas analíticas y grafique su respuesta en MATLAB.

8.8.3 Calcule la integral de las funciones indicadas en el problema 8.8.2 . Considere el intervalo de integración $[0, \pi]$ y el paso de integración de 0.001 segundos. Grafique el resultado en el ambiente de **MATLAB**.



Control

Capítulo



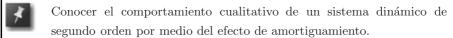
- 9.1 Introducción
- 9.2 Sistemas de segundo orden
- 9.3 Control de temperatura
- 9.4 Resumen
- 9.5 Referencias selectas
- 9.6 Problemas propuestos

Competencias

Implementar algoritmos numéricos de integración y diferenciación, algoritmos de control, sistemas discretos, así como conocer la instrumentación electrónica básica para acoplamiento de sensores con señales de baja magnitud.

Desarrollar habilidades en:







9.1 Introducción



El modelo dinámico de un sistema o proceso representa la descripción completa y detallada de todos sus fenómenos físicos, resulta una herramienta importante para el análisis matemático de estabilidad, diseño de algoritmos de control y es la base fundamental de programación en simulación. Dentro del contexto de simulación, la estructura básica es por medio de una ecuación diferencial ordinaria de primer orden:

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t)),$$
siendo $\boldsymbol{x} \in {\rm I\!R}^n$ la variable fase,

la cual contiene la información necesaria para poder estudiar el comportamiento de la planta, $\dot{x}(t) \in \mathbb{R}^n$ es la variación temporal de la variable de estado (velocidad de movimiento) y $f \in \mathbb{R}^n$ es un mapa vectorial que define las características fundamentales del sistema (lineal y no-lineal). Un caso particular del modelo dinámico $\dot{x}(t) = f(x(t))$ se encuentra el sistema lineal descrito por:

$$\dot{\boldsymbol{x}}(t) = A\boldsymbol{x}(t) + B\boldsymbol{u}(t)$$
$$y(t) = \boldsymbol{c}^T\boldsymbol{x}(t)$$

donde $A \in \mathbb{R}^{n \times n}$, $B \in \mathbb{R}^{n \times 1}$, $\mathbf{c} \in \mathbb{R}^{n \times 1}$

En este capítulo se describen métodos numéricos para calcular derivadas e integrales de funciones y variables de estado, que forma parte de la estructura matemática del algoritmo de control que se ejecuta en la tarjeta Arduino, y cuyo resultado se gráfica en MATLAB.

También se describe el desarrollo de un control PID de temperatura para un horno eléctrico, regla de sintonía para las ganancias del control, instrumentación electrónica sobre el acondicionamiento del sensor de temperatura (termopar) y compensador de temperatura ambiente, etapa de potencia, así como la discusión sobre los detalles de programación.

282 Control



9.2 Sistemas de segundo orden

N sistema dinámico lineal de segundo orden merece atención especial, ya que describe el comportamiento cualitativo y analítico de una gran cantidad de sistemas físicos (plantas reales) como son: sistemas mecánicos (masa-resorte-amortiguador), sistemas eléctricos y electrónicos, algoritmos de control, entre otros más. Representa la base para entender sistemas físicos mucho más complicados.

Debido a que el modelo dinámico de segundo orden representa un sistema físico, para su estudio es necesario tomar en cuenta las condiciones iniciales de dicho sistema. Estas condiciones caracterizan sólo a una función y(t) de la familia de soluciones del sistema.

Considere un sistema lineal de segundo orden representado por la siguiente función de transferencia:

$$\frac{y(s)}{u(s)} = \frac{b_0}{s^2 + a_1 s + a_0} \tag{9.1}$$

donde los coeficientes $b_0, a_0, a_1 \in \mathbb{R}_+$ son números reales constantes positivos, $s \in C$ es un número complejo, y(s) representa la respuesta del sistema y u(s) es la entrada.



Orden de un sistema

El orden de un sistema se refiere al grado del polinomio característico, es decir, el número de polos que tiene la función de transferencia del sistema.

$$G(s) = \frac{y(s)}{u(s)} = \frac{\text{num}(s)}{\text{den}(s)}$$

donde $\operatorname{num}(s)$ y $\operatorname{den}(s)$ son polinomios con coeficientes constantes reales que representan al numerador y denominador, respectivamente.

Desde el punto de vista cualitativo, la respuesta temporal del sistema (9.1) puede presentar varios tipos de comportamiento, por ejemplo transitorios: subamortiguados, sobreamortiguados, amortiguamiento crítico y estados oscilatorios o inclusive crecimiento sostenido de la respuesta (inestabilidad). Generalmente, esta clase de dinámica se define en términos de ciertos parámetros de diseño denominados factor de amortiguamiento ρ y frecuencia natural de resonancia w_n .

Por lo tanto, es útil expresar a la ecuación (9.1) de la siguiente forma:

$$\frac{y(s)}{u(s)} = \frac{w_n^2}{s^2 + 2\rho w_n s + w_n^2} \tag{9.2}$$

El factor ρ determina el efecto de amortiguamiento, el cual se interpreta como un freno mecánico, y es un fenómeno disipativo que convierte la energía mecánica en energía térmica, logrando moldear la respuesta del sistema, decreciendo y eliminado los sobreimpulsos y oscilaciones, obteniendo una respuesta suave si el fenómeno disipativo se incrementa.

La respuesta transitoria de un sistema de segundo orden tiene un conjunto de detalles importantes que dependiendo del factor de amortiguamiento ρ definen el desempeño del sistema.

Una clase de entrada ampliamente estudiada en los sistemas lineales es la entrada escalón unitario, bajo condiciones iniciales cero, la respuesta de un sistema de segundo orden a una entrada de escalón es usualmente utilizada para evaluar las características de la planta de estudio.

Respuesta oscilatoria

Cuando el factor de amortiguamiento $\rho = 0$, los polos de la función de transferencia (9.2) son polos conjugados ubicados sobre el eje imaginario y la parte real es cero. La respuesta es oscilatoria, permaneciendo en este estado en la fase estacionaria.

Respuesta subamortiguada

Si el factor de amortiguamiento ρ satisface la relación $0 < \rho < 1$, los polos de la función de transferencia (9.2) son complejos conjugados con parte real diferente de cero y se ubican en el semiplano izquierdo s, provocando que el estado transitorio tenga varios sobreimpulsos con un comportamiento oscilatorio decreciente.

Por lo tanto, la respuesta temporal y(t) tiene sobreimpulsos debido a componentes senoidales, atenuándose por decaimiento exponencial; la cantidad de sobreimpulsos determinan el valor deseable de la razón de amortiguamiento. Por ejemplo, para $\rho=0.4$, produce un sobretiro del 25.4 %, mientras que $\rho=0.8$ genera un sobreimpulso del 1.8 %. Para algunas aplicaciones este tipo de oscilaciones puede ser tolerado, como el traslado de vasos realizado por un robot manipulador.

Amortiguamiento crítico

Cuando $\rho = 1$, el sistema tiene amortiguamiento crítico, la respuesta en régimen transitoria es suave sin oscilaciones, debido a que no contiene componentes senoidales.

284 Control

Respuesta sobreamortiguada

El tipo de respuesta y(s) es de acción lenta, y no presenta oscilaciones alrededor del punto final. En algunas aplicaciones la ausencia de oscilaciones es necesario, tal es el caso de un elevador donde no se permiten oscilaciones en cada piso. En este caso el factor de amortiguamiento $\rho > 1$.



9.2.1Ecuación en variables de estado

El modelo matemático en variables de estado determina el comportamiento dinámico del sistema a través de una ecuación diferencial de primer orden con la siguiente estructura:

$$\dot{x} = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = Ax + Bu = \underbrace{\begin{bmatrix} 0 & 1 \\ -w_n^2 & -2\rho w_n \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}}_{x} + \underbrace{\begin{bmatrix} 0 \\ 1 \end{bmatrix}}_{B} u \tag{9.3}$$

$$y = c^T x = \underbrace{\begin{bmatrix} 1 & 0 \end{bmatrix}}_{C^T} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$
 (9.4)

donde los parámetros del sistema están representados por: $A \in \mathbb{R}^{2\times 2}$, $B \in \mathbb{R}^{2\times 1}$, $c \in \mathbb{R}^{2\times 1}$, la variable $x \in \mathbb{R}^2$ se denomina variables de estado fase, contiene la información suficiente sobre la descripción dinámica de la planta, la velocidad de movimiento es $\dot{x} \in \mathbb{R}^2$; además, $u \in \mathbb{R}$ es la entrada a la planta y la respuesta del sistema representada por $y \in \mathbb{R}$.

El modelo (9.3) es la base matemática para realizar cualquier tipo de simulación y por lo tanto aprender con profundidad el análisis cualitativo y analítico de los sistemas lineales.

La solución completa de la ecuación (9.3) se obtiene a través del siguiente procedimiento.

Multiplicando en ambos lados de la igualdad (9.3) por la matriz $e^{-At} \in \mathbb{R}^{2\times 2}$:

$$e^{-At}\dot{x}(t) = e^{-At}Ax(t) + e^{-At}Bu(t)$$
 (9.5)

(9.6)

$$\underbrace{e^{-At} \left[\dot{x}(t) - Ax(t) \right]}_{\frac{d}{dt} \left[e^{-At} x(t) \right]} = e^{-At} Bu(t)$$

$$\frac{d}{dt} \left[e^{-At} \boldsymbol{x}(t) \right] = e^{-At} B u(t) \qquad \Rightarrow \qquad d \left[e^{-At} \boldsymbol{x}(t) \right] = e^{-At} B u(t) \tag{9.7}$$

Integrando la ecuación (9.7) sobre el intervalo $t \in [t_0, t]$ se obtiene:

$$\int_{t_0}^t d[e^{-A\sigma} \mathbf{x}(\sigma)] = e^{-At} \mathbf{x}(t) - e^{-At_0} \mathbf{x}(t_0) = \int_{t_0}^t e^{-A\sigma} Bu(\sigma) d\sigma$$
 (9.8)

donde $x(t_0) \in \mathbb{R}^2$ es la condición inicial del sistema.

Despejando la variable de estado x(t) de la ecuación (9.8) se obtiene la siguiente expresión:

$$\mathbf{x}(t) = e^{A(t-t_0)}\mathbf{x}(t_0) + \int_{t_0}^t e^{A(t-\sigma)}Bu(\sigma)d\sigma \quad \forall \ t > t_0$$

$$(9.9)$$

La versión discreta de la ecuación (9.9) se obtiene utilizando un retenedor de orden cero, adquiriendo la siguiente estructura matemática:

$$x(t_k) = e^{A(t_k - t_{k-1})} x(t_{k-1}) + \int_{t_{k-1}}^{t_k} e^{A(t_k - \sigma)} Bu(\sigma) d\sigma$$
(9.10)

donde t_k es el tiempo discreto, el cual se define como múltiplos del periodo de muestreo h, es decir: $t_k = kh$, siendo k un número entero positivo. Además, se satisface $t_{k-1} = (k-1)h$, por lo que: $t_k - t_{k-1} = h$.

Es adquiere la siguiente forma:

$$x(t_k) = e^{Ah}x(t_{k-1}) + \int_0^h e^{A\sigma}Bu(\sigma)d\sigma$$

Una notación más adecuada para la solución discreta $x(t_k)$ es la siguiente:

$$\mathbf{x}(t_k) = \Phi \mathbf{x}(t_{k-1}) + \Gamma u(t_{k-1})$$

$$y(t_k) = \mathbf{c}^T \mathbf{x}(t_k)$$
(9.11)

donde:

$$\Phi = e^{Ah} \in \mathbb{R}^{2 \times 2}$$

$$\Gamma = \int_0^h e^{A\sigma} B d\sigma$$

Por lo tanto, el problema de encontrar la versión discreta de la ecuación (9.3) se reduce a determinar Φ y Γ para obtener el sistema (9.11).

286 Control

& Ejemplo 9.1

Analizar el siguiente sistema dinámico lineal para una entrada u(t) escalón de amplitud unitaria:

$$\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -w_n^2 & -2\rho w_n \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(t)$$

$$y(t) = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix}$$

$$(9.12a)$$

$$y(t) = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} \tag{9.12b}$$

- a) Simular el sistema dinámico usando el método de integración Runge-Kutta 4/5.
- b) Obtener el sistema discreto:

$$\mathbf{x}(t_k) = \Phi \mathbf{x}(t_{k-1}) + \Gamma u(t_{k-1}) \tag{9.13a}$$

$$y(t_k) = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_1(t_k) \\ x_2(t_k) \end{bmatrix}$$
 (9.13b)

- c) Implemente en la tarjeta Arduino el algoritmo (9.13a) y grafique los resultados en MATLAB.
- d) Compare los resultados de simulación por ode45(...) de la ecuación (9.12a) y el sistema discreto 9.13a.

Considere la frecuencia natural de oscilación $w_n = 1$ rad/seg, con un factor de amortiguamiento $\rho = 0.1$ y un periodo de muestreo h=1 mseg.

Solución

En este ejemplo se analiza cualitativamente la respuesta y(t) del sistema de segundo orden (9.13a)-(9.13b) a una entrada escalón de magnitud unitaria u(t), es decir no existe retroalimentación de la salida, en otras palabras, la simulación se considera en lazo abierto, como se indica en la figura 9.1. La finalidad es exhibir el comportamiento subamortiguado de la respuesta del sistema en el régimen transitorio (debido a que el factor de amortiguamiento es $\rho = 0.1 < 1$), es decir, mostrar las componentes oscilatorias y sobreimpulsos que paulatinamente se van atenuando hasta alcanzar el estado estacionario.

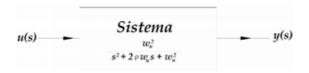


Figura 9.1 Sistema en lazo abierto.

El sistema de segundo orden (9.13a) será simulado mediante el algoritmo de Runge-Kutta 4/5, usando la función de MATLAB ode45(...), dicho resultado será comparado con la versión discreta del mismo sistema. La ecuación del algoritmo discreto (9.13a) será implementada en lenguaje C para su ejecución en una tarjeta electrónica Arduino, enviando las variables de estado a un script o programa en MATLAB para su presentación gráfica y su comparación cualitativa.

La función de transferencia G(s) del sistema está dada como:

$$G(s) = \frac{\text{num}(s)}{\text{den}(s)} = \frac{w_n^2}{s^2 + 2\rho w_n s + w_n^2} = \frac{1}{s^2 + 0.2s + 1}$$
(9.14)

debido a que el factor de amortiguamiento ρ es menor a uno, entonces la función de transferencia G(s) tiene dos polos complejos conjugados, los cuales se obtienen mediante programación. En la ventana de comandos de **MATLAB** se obtienen los polos de la función de transferencia G(s) (ceros del polinomio denominador den(s)):

fx>> format short $\,\hookleftarrow\,$

fx >> den=[1, 0.2, 1];%coeficientes del denominador $den(s) = s^2 + 0.2s + 1 \iff$

fx>>r=roots(den)%raíces o ceros del polinomio denominador. \hookleftarrow

r=
$$0.1 + 0.995i$$

 $-0.1 - 0.995i$

Otra forma de obtener el mismo resultado es a través del cálculo de los valores propios de la matriz A del sistema (9.12a), por medio de las siguientes instrucciones:

fx >> format short $\ensuremath{\hookleftarrow}$

$$fx >> A=[0, 1; -1, -0.2];$$
%inicializa $A=\begin{bmatrix} 0 & 1 \\ -1 & -0.2 \end{bmatrix}$. \longleftrightarrow

 $fx>> \operatorname{eig}(\mathbf{A})$ %
valores propios de la matriz $A\in \mathbb{R}^{2\times 2}$. \hookleftarrow

ans=
$$-0.1 + 0.995i$$

 $-0.1 - 0.995i$

Debido a que la función de transferencia G(s) (9.14) tiene dos polos complejos conjugados (valores propios de la matriz A), el procedimiento para obtener los coeficientes α_0 y α_1 que forman a la matriz $\Phi \in \mathbb{R}^{2\times 2}$ consta de los siguientes pasos:

$$e^{(-0.1 + 0.995i)h} = \alpha_0 + (-0.1 + 0.995i) \alpha_1$$
 (9.15a)

$$e^{-(0.1+0.995i)h} = \alpha_0 - (0.1+0.995i)\alpha_1$$
 (9.15b)

Resolviendo el sistema de ecuaciones (9.15a)-(9.15b):



Multiplicando por -1 a la ecuación (9.15b) y substrayendo con la ecuación (9.15a) se obtienen las siguientes operaciones:

$$e^{(-0.1 + 0.995i)h} = \alpha + (-0.1 + 0.995i)\alpha_1$$
 (9.16a)

$$-e^{-(0.1+0.995i)h} = -\alpha_0 + (0.1+0.995i)\alpha_1$$
 (9.16b)

$$e^{-0.1h} \left[e^{0.995i h} - e^{-0.995i h} \right] = i2 (0.995) \alpha_1$$
 (9.16c)

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

FERNANDO REYES CORTÉS • JAIME CID MONJARAZ

Alfaomega

288 Control



Tomando en cuenta que $e^{0.995i} = \cos(0.995h) + i \sin(0.995h)$ y $e^{-0.995i} = \cos(0.995h) - \sin(0.995h)$, se obtiene:

$$e^{-0.1h} \left[\cos(0.995 h) + i \sin(0.995 h) - \cos(0.995 h) + i \sin(0.995 h) \right] = i 2 (0.995) \alpha_1$$
 (9.17)

$$/2e^{-0.1h} \operatorname{sen}(0.995 h) = /2 (0.995) \alpha_1$$
 (9.18)

El valor para α_1 es:

$$\alpha_1 = e^{-0.1h} \frac{\sin(0.995 \ h)}{0.995} \tag{9.19}$$



Para encontrar α_0 se sustituye α_1 (9.19) en la ecuación (9.15a):

$$\alpha_{0} = e^{(-0.1 + 0.995i)h} + 0.1\alpha_{1} - 0.995\alpha_{1} i$$

$$= e^{-0.1h} \left[\cos(0.995h) + i \sin(0.995h) \right] + 0.1e^{-0.1} \frac{\sin(0.995h)}{0.995}$$

$$-0.995 ie^{-0.1} \frac{\sin(0.995h)}{0.995}$$

$$= e^{-0.1h} \left[\cos(0.995h) + i \sin(0.995h) \right] + 0.1e^{-0.1} \frac{\sin(0.995h)}{0.995}$$

$$-ie^{-0.1} \sin(0.995h)$$

$$= e^{-0.1h} \cos(0.995h) + ie^{-0.1h} \sin(0.995h) + 0.1e^{-0.1} \frac{\sin(0.995h)}{0.995}$$

$$-ie^{-0.1} \sin(0.995h)$$

$$\alpha_{0} = e^{-0.1h} \left[\cos(0.995h) + 0.1 \frac{\sin(0.995h)}{0.995} \right]$$

$$(9.21)$$

Por lo tanto, la matriz $\Phi = [\alpha_0 I + \alpha_1 A] \in \mathbb{R}^{2 \times 2}$ toma la siguiente forma:

$$\Phi = \alpha_0 \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + \alpha_1 \begin{bmatrix} 0 & 1 \\ -1 & -0.2 \end{bmatrix} = \begin{bmatrix} \alpha_0 & 0 \\ 0 & \alpha_0 \end{bmatrix} + \begin{bmatrix} 0 & \alpha_1 \\ -\alpha_1 & -0.2\alpha_1 \end{bmatrix} = \begin{bmatrix} \alpha_0 & \alpha_1 \\ -\alpha_1 & \alpha_0 - 0.2\alpha_1 \end{bmatrix}$$

$$= \begin{bmatrix} e^{-0.1h} \begin{bmatrix} \cos(0.995h) + 0.1 \frac{\sin(0.995h)}{0.995} \end{bmatrix} e^{-0.1h} \underbrace{\begin{bmatrix} \cos(0.995h) + 0.1 \frac{\sin(0.995h)}{0.995} \\ e^{-0.1h} \underbrace{\begin{bmatrix} \cos(0.995h) + 0.1 \frac{\sin(0.995h)}{0.995} \end{bmatrix}}_{0.995} \end{bmatrix} e^{-0.1h} \underbrace{\begin{bmatrix} \cos(0.995h) + 0.1 \frac{\sin(0.995h)}{0.995} \\ -e^{-0.1h} \underbrace{\begin{bmatrix} \cos(0.995h) + 0.1 \frac{\sin(0.995h)}{0.995} \end{bmatrix}}_{0.995} e^{-0.1h} \underbrace{\begin{bmatrix} e^{-0.1h} \underbrace{\sin(0.995h)}_{0.995} \\ 0.995 \end{bmatrix}}_{0.995} \end{bmatrix} }_{e^{-0.1h} \underbrace{\begin{bmatrix} \cos(0.995h) + 0.1 \frac{\sin(0.995h)}{0.995} \end{bmatrix}}_{0.995} \end{bmatrix}$$

$$(9.23)$$

Resolviendo la expresión para $\Gamma = \int_0^h e^{Ah} B dh$ conduce a:

$$\Gamma = \begin{bmatrix} 0.9999750006 - 0.005025000003e^{-0.1h} [199.0\cos(0.995h) + 20.0\sin(0.995h)] \\ -e^{-0.1h} [3.1 \times 10^{-13}\cos(0.995h) - 1.005025126\sin(0.995h)] \end{bmatrix}$$
(9.24)

El sistema discreto del sistema dinámico continuo (9.12a) toma la siguiente forma:

$$\begin{bmatrix} x_1(t_k) \\ x_2(t_k) \end{bmatrix} = \begin{bmatrix} e^{-0.1h} \left[\cos(0.995h) + 0.1 \frac{\sin(0.995h)}{0.995} \right] & e^{-0.1h} \frac{\sin(0.995h)}{0.995} \\ -e^{-0.1h} \frac{\sin(0.995h)}{0.995} & e^{-0.1h} \left[\cos(0.995h) - 0.1 \frac{\sin(0.995h)}{0.995} \right] \end{bmatrix} \begin{bmatrix} x_1(t_{k-1}) \\ x_2(t_{k-1}) \end{bmatrix} \\
+ \begin{bmatrix} 0.9999750006 - 0.005025000003e^{-0.1h} \left[199.0 \cos(0.995h) + 20.0 \sin(0.995h) \right] \\ -e^{-0.1h} \left[3.1 \times 10^{-13} \cos(0.995h) - 1.005025126 \sin(0.995h) \right] \end{bmatrix} u(t_{k-1}) \quad (9.25)$$

Alfaomega

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

Fernando Reyes Cortés • Jaime Cid Monjaraz

Observe que una vez seleccionado el valor del periodo de muestreo h, las matrices Φ (9.23) y Γ (9.24) se convierten en constantes, por ejemplo si h=0.001 segundos, entonces se obtiene la siguiente expresión:

$$\begin{bmatrix} x_1(t_k) \\ x_2(t_k) \end{bmatrix} = \begin{bmatrix} 1 & 0.0009999 \\ -0.0009999 & 0.9998 \end{bmatrix} \begin{bmatrix} x_1(t_{k-1}) \\ x_2(t_{k-1}) \end{bmatrix} + \begin{bmatrix} 4.9997 \times 10^{-7} \\ 0.0009999 \end{bmatrix} u(t_{k-1})$$
(9.26)

Para cada valor que tome el periodo de muestreo h, se obtendrá una expresión discreta diferente.

El cuadro de código Arduino 9.1 describe el sketch **cap9_sistema2dok**, contiene el algoritmo discreto (9.25) y envía información para su desplegado gráfico a **MATLAB**. Se emplea la librería de funciones matemáticas en el header o cabecera (línea 1); las variables globales se encuentran declaradas de las líneas 2 a la 7. Particularmente, la variable **base_tiempo** contiene el tiempo en segundos para ejecutar el algoritmo discreto sobre la tarjeta Arduino. La subrutina que envía a **MATLAB** la información del algoritmo como el tiempo discreto t_k y las variables de estado (posición $x_1(t_k)$ y velocidad $x_2(t_k)$) se encuentra en la línea 8.



Ejemplos de sistemas discretos

En el sitio Web de este libro, se describen ejemplos para discretizar el sistema dinámico de segundo orden, con polos de la función de transferencia corresponden a los siguientes casos: oscilador ($\rho=0$), subamortiguado ($\rho<1$), amortiguamiento crítico ($\rho=1$) y sobreamortiguado ($\rho>1$). Se presenta una comparación usando simulación por Runge-Kutta 4/5 del sistema $\dot{\boldsymbol{x}}=A\boldsymbol{x}+B\boldsymbol{u}$ con la función ode45(...) de MATLAB y la versión discreta $\boldsymbol{x}(t_k)=\Phi\boldsymbol{x}(t_{k-1})+\Gamma\boldsymbol{u}(t_{k-1})$ que se ejecuta en la tarjeta Arduino.

La rutina de configuración **setup()** se ubica en la línea 15, se programa la velocidad de comunicación serial en 19200 Baudios entre la computadora y la tarjeta Arduino. Dependiendo de las características de la computadora donde se encuentran los ambientes de programación Arduino y **MATLAB**, el usuario puede configurar la velocidad de comunicación en: 9600, 14400, 19200, 28800, 38400, 57600 y 115200.

La programación que permite el adecuado enlace de comunicación USB entre la tarjeta Arduino y el entorno de \mathbf{MATLAB} se ubica de las líneas 19 a la 54; la variable $\mathbf{bandera}$ tiene el valor booleano verdadero, el procedimiento consiste en que la tarjeta Arduino envía el comando inicial 33.0000; también se transmite información del periodo de muestreo h y el número de puntos a graficar (en este ejemplo son 50,000 puntos). El script en \mathbf{MATLAB} verifica el comando inicial, posteriormente Arduino queda en espera de recibir desde \mathbf{MATLAB} el comando de respuesta 95. De esta forma, cualquiera de los dos programas se puede ejecutar, sin importar el orden. La

290 Control

figura 9.2 muestra el diagrama de flujo para sincronizar el enlace de comunicación por puerto USB, permitiendo la correcta comunicación de transmisión/recepción de datos para su representación gráfica. Es importante aclarar que el conjunto de datos enviados (tiempo discreto, posición y velocidad) desde la tarjeta Arduino hacia el ambiente de programación MATLAB utilizando como medio de comunicación el puerto USB para su representación gráfica no es en tiempo real.

Durante la fase de sincronía entre el ambiente de programación en MATLAB y la tarjeta Arduino, se registran las condiciones iniciales, así como el cálculo de los parámetros α_0 , α_1 , matrices Φ y Γ ; estos cálculos se realizan una sola vez y posteriormente permanecen como valores constantes. A partir de la línea 55 se ejecuta el algoritmo discreto que permite obtener los valores de $x(t_k) = \Phi x(t_{k-1}) + \Gamma u(t_{k-1})$. En cada iteración, se envían para su representación gráfica en MATLAB los k-ésimos datos del tiempo discreto t_k , posición $x_1(t_k)$ y velocidad $x_2(t_k)$ (ver línea 63). La actualización de las variables de estados en tiempo discreto anterior (t_{k-1}) se realiza en la línea 66, así como la limpieza de registros que intervienen en el cálculo aritmético del proceso recursivo (línea 68). El número de iteraciones se calcula como el valor de la variable base tiempo (ver línea 5) dividida entre el periodo de muestreo h, tal y como se transmite este valor en la fase inicial de sincronía, para este ejemplo 50,000 puntos (línea 38).

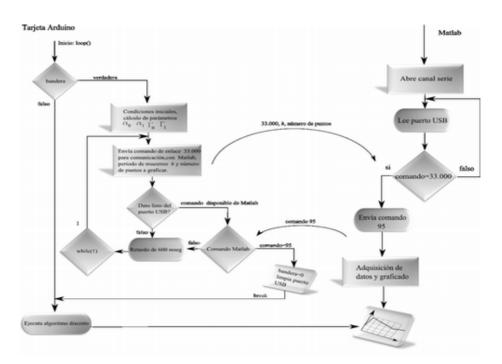


Figura 9.2 Enlace de comunicación USB entre la tarjeta Arduino y MATLAB.

```
€ Código Arduino 9.1: sketch cap9_sistema2dok
   Arduino. Aplicaciones en Robótica y Mecatrónica.
   Capítulo 9 Control.
   Fernando Reyes Cortés y Jaime Cid Monjaraz.
   Alfaomega Grupo Editor: "Te acerca al conocimiento".
    Sketch cap9_sistema2dok.ino
 1 #include <math.h>// librería para funciones matemáticas.
 2 float tk=0; //tiempo discreto.
 3 float h=0.001;//periodo de muestreo de un milisegundo.
 4 int i, j, fase=0, bandera=1;//variable bandera funciona como semáforo para enlace USB.
 5 float base_tiempo=50,u;//50000=base_tiempo/h, puntos a graficar en MATLAB.
 6 float Phi[2][2], Gamma[2][1];
 7 float xk[2][1], xka[2][1];
 8 void envia_datosMatlab(float t, float var1, float var2){//envía a MATLAB
    información.
        Serial.print(t);//tiempo discreto t_k.
        Serial.print(" ");//espacio en blanco para separar datos.
10
        Serial.print(var1);//variable de estado (posición x_1(t_k)).
11
        Serial.print(" ");//inserta espacio en blanco para separar datos.
12
        Serial.println(var2);//variable de estado (velocidad x_2(t_k)).
13
14 }
15 void setup() {//rutina de configuración.
        Serial.begin(19200);//19200, 28800, 38400, 57600 Baudios.
16
17 }
18 void loop(){
19
        if (bandera){//fase inicial de sincronía para comunicación serial USB.
           //Cálculo de la constante: \Gamma = \begin{bmatrix} \gamma_{00} \\ \gamma_{10} \end{bmatrix} \in \mathbb{R}^{2 \times 1}.
20
           Gamma[0][0] = 0.9999750006 - 0.005025000003*exp(-0.1*h)*(199.0*cos(0.995*h) + 20.0*sin(0.995*h));
21
22
           Gamma[1][0] = -\exp(-0.1*h)*(3.1*(10e-13)*\cos(0.995*h) - 1.005025126*\sin(0.995*h)); //\gamma_{10}.
           //Cálculo de la constante: \Phi = \begin{bmatrix} \phi_{00} & \phi_{01} \\ \phi_{10} & \phi_{11} \end{bmatrix} \in \mathbb{R}^{2 \times 2}.
23
           Phi[0][0]=exp(-0.1*h)*(cos(0.995*h)+0.1*sin(0.995*h)/0.995);//\phi_{00}.
24
           Phi[0][1] = exp(-0.1*h)*sin(0.995*h)/0.995;//\phi_{01}.
25
           Phi[1][0]=-exp(-0.1*h)*sin(0.995*h)/0.995;//\phi_{10}.
26
           Phi[1][1]=exp(-0.1*h)*(cos(0.995*h)-0.1*sin(0.995*h)/0.995);//\phi_{11}.
27
```

```
Continúa código Arduino 9.1a: sketch cap9_sistema2dok
   Arduino. Aplicaciones en Robótica y Mecatrónica.
   Capítulo 9 Control.
   Fernando Reyes Cortés y Jaime Cid Monjaraz.
   Alfaomega Grupo Editor: "Te acerca al conocimiento".
   Continuación del sketch cap9_sistema2dok.ino
28
          tk=0;// tiempo discreto t_k inicia en cero segundos.
          u=0;//función escalón u(0) = 0, si t_k = 0.
29
          //Condición inicial del vector de estados: \boldsymbol{x}(0) = \begin{bmatrix} x_{00}(0) \\ x_{10}(0) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.
30
          xk[0][0]=xka[0][0]=0;//x_{00}(0)=0, xka significa estado anterior x_{00}(t_{k-1}).
31
          xk[1][0]=xka[1][0]=0;//x_{10}(0)=0, estado anterior x_{10}(t_{k-1}).
32
          do{//protocolo de comunicación y enlace entre Arduino y MATLAB.
33
              Serial.print(33.0000,4);//envía comando de inicio para MATLAB.
34
              Serial.print(" ");//separación de datos.
35
36
              Serial.print(h,4);//periodo de muestreo h.
              Serial.print(" ");//separación de datos.
37
              Serial.println(base_tiempo/h,4);//intervalo de tiempo a graficar en MATLAB.
38
              if(Serial.available()>0){//respuesta del script MATLAB.
39
                 fase=Serial.read();//recibe comando de respuesta.
40
                 if (fase=95){
41
                     bandera=0;//limpia semáforo de sincronía.
42
                     //Envía datos de limpieza del puerto serial USB.
43
                     Serial.print(0);
44
                     Serial.print(" ");
45
                     Serial.print(0);
46
                     Serial.print(" ");
47
                     Serial.println(0);
48
                     break;
49
50
              }
51
              delay(600);//tiempo de retardo para ajuste de sincronía y enlace USB.
52
          \}while(1);
53
54
```

```
Continúa código Arduino 9.1b: sketch cap9_sistema2dok
  Arduino. Aplicaciones en Robótica y Mecatrónica.
  Capítulo 9 Control.
  Fernando Reyes Cortés y Jaime Cid Monjaraz.
  Alfaomega Grupo Editor: "Te acerca al conocimiento".
   Continuación del sketch cap9_sistema2dok.ino
55
       if(tk<base_tiempo){//tiempo de ejecución en la tarjeta Arduino.
          if(tk>0) u=1;//señal de entrada escalón de magnitud unitaria u(t_k).
56
          for(i=0; i<2; i++){//algoritmo discreto: \boldsymbol{x}(t_k) = \Phi \boldsymbol{x}(t_{k-1}) + \Gamma u(t_{k-1}).
57
             for(j=0; j<2; j++){//algoritmo para calcular la versión recursiva de x(t_k).
58
                 xk[i][0]=Phi[i][j]*xka[j][0]+xk[i][0];
59
             }
60
             xk[i][0]=xk[i][0]+Gamma[i][0]*u;//complementa cálculo recursivo de <math>x(t_k).
61
62
          envia_datosMatlab(tk,xk[0][0],xk[1][0]);//envía datos a graficar en MATLAB.
63
64
          tk=tk+h;//evolución del tiempo discreto como múltiplos de h.
65
          delay(h*1000);//retardo de un milisegundo.
          xka[0][0]=xk[0][0];//estado anterior x_{00}(t_{k-1})=x_{00}(t_k).
66
          xka[1][0]=xk[1][0];//estado anterior <math>x_{10}(t_{k-1})=x_{10}(t_k).
67
          xk[0][0]=xk[1][0]=0;//limpia registros de cálculos aritméticos en la línea 59.
68
       }// fin de la instrucción if(tk<base_tiempo){...}, línea 55.
69
70 }//fin de la subrutina principal loop(){...}, línea 18.
```

El cuadro de código 9.2 describe al script **cap9_simusistema2doA**, este programa representa la interface de comunicación gráfica de las variables de interés al usuario sobre el algoritmo (sketch **cap9_sistema2dok**) que se ejecuta en la tarjeta Arduino.

En la línea 3 se programa la velocidad de comunicación serial (con el mismo valor en Baudios utilizado en la línea 16 del cuadro de código Arduino 9.1), también se define el puerto USB (en este caso COM9), el cual depende de los recursos y características de la computadora.

El puerto serial que utiliza **MATLAB** deberá ser el mismo que se utiliza por la tarjeta Arduino, para mayores detalles consultar la sección 8.2, figura 8.1, página 250. La comunicación USB queda abierta en la línea 4. La fase de sincronía entre la tarjeta Arduino y **MATLAB** se encuentra programada de la línea 9 a la 17.

```
A Código MATLAB 9.2 cap9_simusistema2doA.m
   Arduino. Aplicaciones en Robótica y Mecatrónica.
   Capítulo 9 Control.
   Fernando Reyes Cortés y Jaime Cid Monjaraz.
   Alfaomega Grupo Editor: "Te acerca al conocimiento".
   Archivo cap9_simusistema2doA.m
                                                           Versión de Matlab 2014a
 1 clc; clear all; close all;
 2 format short g
 3 canal_serie = serial('COM9', 'BaudRate', 19200, 'Terminator', 'CR/LF'); %crear objeto serial.
 4 fopen(canal_serie); %abrir puerto de comunicación serial.
 5 xlabel('Segundos'); ylabel('Datos'); title('Control de procesos utilizando Arduino');
 6 grid on; hold on;
 7 \text{ fase}=0;
 8 disp('Sincronizando enlace con tarjeta Arduino' )
 9 while (1)
       fase=fscanf(canal_serie, '%f %f %f ',[3,1]); %lee el puerto serie.
       disp(['Comando=', num2str(fase(1,1)), 'Periodo de muestreo=',...
11
       num2str(fase(2,1)), 'Número de puntos a graficar=', num2str(fase(3,1))]);
12
       if (fase(1,1)==33.0000)
13
           disp('Inicia adquisición de datos con la tarjeta Arduino')
14
15
           break;
       end
16
17 end
18 fwrite(canal_serie, 95, 'int');
19 prop = line(nan, nan, 'Color', 'b', 'LineWidth', 1);
20 fscanf(canal_serie, '%f %f %f ',[3,1]);%basura en el puerto serial.
21 disp('Graficando variables Arduino...');
22 for i=1:(fase(3,1))
       datos = fscanf(canal_serie, '%f %f %f ',[3,1]); %leer el puerto serie.
23
       tiempo(i,1) = datos(1,1);
24
       var1(i,1) = datos(2,1); var2(i,1) = datos(3,1);
       set(prop, 'YData', var1(1:i,1), 'XData', tiempo(1:i,1));
26
27
       drawnow;
28 end
```

En la línea 18 se envía el comando de respuesta MATLAB (comando 95) a la tarjeta Arduino para iniciar la ejecución del algoritmo discreto. Es clave limpiar el puerto serial de posibles datos erróneos, esto se lleva a cabo con una lectura al puerto USB como se indica en la línea 20.

El tiempo discreto y las variables de estado posición y velocidad se registran en tiempo, var1 y var2, respectivamente (ver líneas 22 a la 28); la presentación gráfica de los 50,000 puntos para var1 se realiza en función del tiempo (no corresponde a tiempo real).

A partir de la línea 29 se cierra la comunicación serial. En la línea 32 se inicia la etapa de simulación del sistema dinámico 9.12a, el tiempo inicial en cero, el periodo de muestreo idéntico al utilizado en la tarjeta Arduino (el cual fue recibido en la línea 10) y el tiempo de simulación corresponde al número de puntos o iteraciones realizadas por el algoritmo discreto (línea 34, cincuenta mil puntos para este ejemplo). También se utilizan las mismas condiciones iniciales del algoritmo discreto (línea 36).



Continúa código MATLAB 9.2a: cap9_simusistema2doA.m

Arduino. Aplicaciones en Robótica y Mecatrónica.

Capítulo 9 Control.

Fernando Reyes Cortés y Jaime Cid Monjaraz.

Alfaomega Grupo Editor: "Te acerca al conocimiento".

Continuación de cap9_simusistema2doA.m

- 29 fclose(canal_serie); %cierra objeto serial.
- 30 delete(canal_serie); %libera memoria asignado al canal serial.
- 31 clear canal_serie; %limpia la variable asignada al canal serial.
- **32** ti=0;
- 33 h=fase(2,1); %periodo de muestreo.
- 34 tf =fase(2,1)*fase(3,1);%tiempo de simulación.
- 35 t=ti:h:tf; %vector tiempo.
- **36** condiciones_iniciales=[0; 0]; %condiciones iniciales.
- 37 opciones=odeset('RelTol', 1e-06, 'InitialStep', h, 'MaxStep', h);
- 38 % solución numérica del sistema dinámico en variables fase $\dot{x}(t) = Ax(t) + Bu$.
- 39 [t,x]=ode45('cap9_sistema2doA',t,condiciones_iniciales,opciones);
- **40** figure
- 41 plot(t, x(:,1), tiempo, var1, t, x(:,2), tiempo, var2)

```
A Código MATLAB 9.3 cap9_sistema2doA.m
  Arduino. Aplicaciones en Robótica y Mecatrónica.
  Capítulo 9 Control.
  Fernando Reyes Cortés y Jaime Cid Monjaraz.
  Alfaomega Grupo Editor: "Te acerca al conocimiento".
                                                       Versión de Matlab 2014a
   Archivo cap9_sistema2doA.m
1 function xp=cap9_sistema2doA(t,x)
      wn=1; % frecuencia natural de oscilación w_n.
2
      rho=0.1; % factor de amortiguamiento \rho.
3
      A = [0 \ 1;
4
          -wn*wn -2*rho*wn;
5
      B = [0;
6
7
          1];
      u(t==0)=0;
8
      u(t>0)=+1;
9
      xp=A*x+B*u; \%\dot{x}=Ax+Bu.
11 end
```

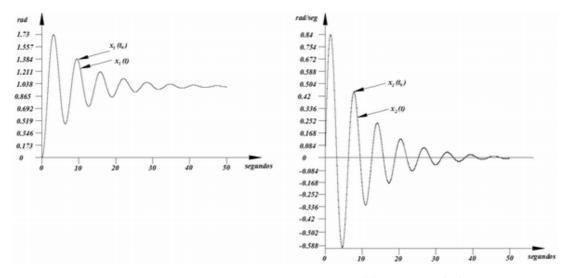


Figura 9.3 Comparación entre la posición $x_1(t)$, vs $x_1(t_k)$ y velocidad $x_2(t)$, vs $x_2(t_k)$.

La línea 37 contiene la configuración de la función **ode45(...)**, observe que para el método de integración numérica Runge-Kutta se establece un error relativo de tolerancia de una millonésima,

así como el paso inicial y el máximo paso de integración igual al periodo de muestreo h.

La solución numérica del sistema dinámico $\dot{x}(t) = Ax(t) + Bu$ se lleva a cabo en la línea 39, retornando el resultado de simulación en las variables \mathbf{t} (tiempo) y en el arreglo $\mathbf{x} \in \mathbb{R}^{50,000 \times 2}$; la primera columna representa la posición y la segunda columna se refiere a la velocidad.

El código 9.3 presenta la implementación del sistema dinámico $\dot{x}(t) = Ax(t) + Bu$ contenido en la función cap9_sistema2doA, es decir:

$$\begin{bmatrix} \dot{x_1}(t) \\ \dot{x}_2(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & -0.2 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(t)$$
 (9.27)

Por notación, en el cuadro de código Arduino 9.1 la variable $\mathbf{xk}[0][0]$ representa la posición del sistema en la k-ésima muestra $x_1(t_k)$, la posición en la muestra anterior es $\mathbf{xka}[0][0]$ o $x_1(t_{k-1})$, la velocidad está denotada por $\mathbf{xk}[1][0]$, es decir $x_2(t_k)$ y la velocidad en la muestra anterior es: $\mathbf{xka}[1][0]$, equivalente a $x_2(t_{k-1})$.

Debido a la sintaxis del lenguaje de MATLAB, el archivo que contiene a la función cap9_sistema2doA con el código correspondiente del sistema dinámico (9.27) deberá llevar el mismo nombre y con extensión .m, es decir: cap9_sistema2doA.m.

Es recomendable que el archivo del programa principal **cap9_simusistema2doA.m**, así como el de la función del sistema dinámico (9.27) **cap9_sistema2doA.m** se encuentren localizados en la misma carpeta o directorio del usuario. De no ser así, cuando el programa principal está en un directorio diferente al de la función del sistema dinámico, se puede utilizar el siguiente código en el cuadro 9.2a, línea 39:

[t,x]=ode45('c:\Arduino\Simulacion\cap9_sistema2doA',t,condiciones_iniciales,opciones);

donde la trayectoria 'c:\Arduino\Simulacion\cap9_sistema2doA' es un ejemplo específico que lo define el usuario.

La comparación de resultados entre el algoritmo recursivo $x(t_k) = \Phi x(t_{k-1}) + \Gamma u(t_{k-1})$ que se ejecuta en la tarjeta Arduino y el obtenido por integración numérica Runge-Kutta del sistema dinámico $\dot{x}(t) = Ax(t) + Bu$ (línea 39) se muestra en la figura 9.3 (ver línea 41). Observe que ambas gráficas coinciden plenamente, ya que la posición y velocidad para $x_1(t), x_2(t)$ se superponen con las señales discretas que se ejecutan en la tarjeta Arduino $x_1(t), x_2(t)$. Observe también que la respuesta del sistema $y(t_k) = x_1(t_k)$.

Debido a que el factor de amortiguamiento $\rho = 0.1 < 1$, el sistema (9.12a) está subamortiguado, como se muestra en la figura 9.3, observe que la respuesta transitoria tiene comportamiento

oscilatorio, acompañada con sobreimpulsos que se van atenuando a medida que el tiempo evoluciona. De hecho, el tiempo para alcanzar la fase estacionaria es mayor a 50 segundos, permaneciendo con oscilaciones sostenidas y pequeñas componentes de rizo.

Hay que recalcar que este tipo de comportamiento es en lazo abierto. Para evitar este tipo de respuesta transitoria es necesario incrementar el freno mecánico a través de la incorporación de un elemento de amortiguamiento en lazo cerrado, a través de un esquema de control como el que se analiza en el ejemplo 9.2.

Los pasos requeridos para ejecutar el sketch **cap9_sistema2dok** en cualquiera de los modelos de las tarjetas Arduino son los que se describen en la sección 2.4 del capítulo 2 **Instalación y puesta a punto del sistema Arduino**. Sin embargo, a continuación se describen en forma resumida:

- Editar el código fuente en lenguaje C del sketch.
- En el menú **Herramientas** del ambiente de programación Arduino seleccionar modelo de tarjeta y velocidad de comunicación serial en 19200 Baudios.
- Compilar el sketch mediante el icono 🕢.
- Descargar el código de máquina a la tarjeta Arduino usando 👈.
- Editar los scripts **cap9_simusistema2doA** y **cap9_sistema2doA** dentro del editor de **MATLAB**.
- Ubicando dentro de la ventana de edición del script **cap9_simusistema2doA** oprima el icono **Run** o la tecla F5.

♣ ♣ Ejemplo 9.2

Analizar el comportamiento del sistema dinámico (9.12a) con un esquema de retroalimentación usando la siguiente ley de control no lineal u(t):

$$u(t) = k_p \tanh(\tilde{x}(t)) - k_v \tanh(x_2(t))$$
(9.28)

donde k_p y k_v representan las ganancias proporcional y derivativa, $\tilde{x}(t)$ es el error de posición definido como la diferencia entre el estado deseado x_d y el estado retroalimentado $x_1(t)$, es decir: $\tilde{x}(t) = x_d - x_1(t)$.

- a) Simular el sistema dinámico resultante en lazo cerrado usando el método de integración Runge-Kutta 4/5.
- b) Implementar en la tarjeta Arduino el sistema discreto (9.13a) con la ley de control (9.28); grafique los resultados en MATLAB.
- c) Compare los resultados de simulación por ode45(...) de la ecuación en lazo cerrado y el correspondiente sistema discreto.

Considere la frecuencia natural de oscilación $w_n=1$ rad/seg, con un factor de amortiguamiento $\rho=0.1$, periodo de muestreo h=1 mseg, estado deseado $x_d=1$ rad, $k_p=20, k_v=20$.

Solución

En el ejemplo 9.1 se presentó el análisis cualitativo del sistema dinámico (9.12a)-(9.12b) en lazo abierto para una entrada escalón de magnitud unitaria. En este ejemplo, el mismo sistema dinámico es analizado de igual forma desde el punto de vista cualitativo bajo el enfoque de retroalimentación de estados.

La ecuación en lazo cerrado que combina el modelo dinámico (9.12a) y el esquema de control no lineal (9.28) resulta una ecuación diferencial no lineal de primer orden con la siguiente estructura:

$$\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \end{bmatrix} = \begin{bmatrix} x_2(t) \\ k_p \tanh(x_d - x_1(t)) - k_v \tanh(x_2(t)) - 0.2x_2(t) - x_1(t) \end{bmatrix}$$
(9.29)

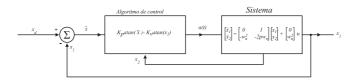


Figura 9.4 Lazo de control.

La figura 9.4 muestra el diagrama a bloques del sistema dinámico (9.12a) con el algoritmo de control (9.28). Note que a pesar de que la planta o sistema dinámico es lineal, la ecuación en lazo cerrado resultante es un sistema dinámico

no lineal, por lo que no aplican los conceptos de función de transferencia, polos o ceros del sistema.

El esquema de control (9.28) utiliza una función saturada del error de posición $\tilde{x}(t)$ del tipo tangente hiperbólica; observe que también incorpora una acción de control derivativa saturada, es decir, la clase de amortiguador que incluye es no lineal, este amortiguador se obtiene por medio de inyección de la velocidad $x_2(t)$ y se suma al fenómeno natural de amortiguamiento (fricción viscosa del sistema $0.2x_2(t)$), como se aprecia en la ecuación en lazo cerrado (9.29).

El cuadro de código Arduino 9.4 describe al sketch **cap9_sistema2dokC** con la programación necesaria para implementar el sistema discreto con el algoritmo de control con retroalimentación de estados. Dicha programación es muy similar al que se presenta en el cuadro 9.1, con la diferencia principal que en las líneas 63-65 del cuadro 9.4b se encuentra la programación del esquema de control (9.28).

El tiempo de ejecución en la tarjeta Arduino corresponde a 10 segundos (ver línea 5), con un periodo de muestreo h de un milisegundo, es decir se procesan diez mil iteraciones.

Por otro lado, el script o programa en MATLAB que se utiliza en esta aplicación corresponde al descrito en el cuadro 9.2a, es decir: cap9_simusistema2doA. Sin embargo, hay que hacer un solo cambio en la línea 39 utilizando el siguiente código:

$[t,x] = ode45 (\ 'cap9_sistema2doCC'\ ,t,condiciones_iniciales,opciones);$

puesto que el archivo **cap9_sistema2doCC.m** (ver cuadro 9.4) contiene la programación del sistema dinámico (9.12a) con el algoritmo de control no lineal (9.28).

Para propósitos de comparar resultados con el sistema discreto que se ejecuta en Arduino, la ecuación diferencial no lineal de primer orden es la que se debe ejecutar en lugar del sistema en lazo abierto 'cap9_sistema2doA' (cuadro 9.3).



Simulación de sistemas dinámicos

En el sitio Web, se desarrollan programas en MATLAB (scripts) que permiten la simulación de sistemas dinámicos lineales y no lineales, empleando la función ode45(...).

```
Código Arduino 9.4: sketch cap9_sistema2dokC
   Arduino. Aplicaciones en Robótica y Mecatrónica.
   Capítulo 9 Control.
   Fernando Reyes Cortés y Jaime Cid Monjaraz.
   Alfaomega Grupo Editor: "Te acerca al conocimiento".
    Sketch cap9_sistema2dokC.ino
 1 #include <math.h>
 2 float tk=0; //condición inicial de la variable temporal discreta.
 3 float h=0.001;//periodo de muestreo.
 4 int i,j, fase=0, bandera=1;
 5 float base_tiempo=10,u;
 6 float Phi[2][2], Gamma[2][1];
 7 float xk[2][1], xka[2][1];
 8 float kp=20, kv=20, xd=1, xtilde;
 9 void envia_datosMatlab(float t, float var1, float var2){
         Serial.print(t);
10
         Serial.print(" ");
        Serial.print(var1);
12
         Serial.print(" ");
13
14
         Serial.println(var2);
15 }
16 void setup() {//Rutina de configuración.
         Serial.begin(19200);//Comunicación serie en 9600 Baudios.
17
18 }
19 void loop(){
         if (bandera){
20
            //Cálculo de la constante: \Gamma = \begin{bmatrix} \gamma_{00} \\ \gamma_{10} \end{bmatrix} \in \mathbb{R}^{2 \times 1}.
21
22
            \operatorname{Gamma}[0][0] = 0.9999750006 - 0.005025000003*\exp(-0.1*h)*(199.0*\cos(0.995*h) + 20.0*\sin(0.995*h));
23
            \operatorname{Gamma}[1][0] = \exp(-0.1*h)*(3.1*(10e-13)*\cos(0.995*h) - 1.005025126*\sin(0.995*h)); \\ \gamma_{10}.
            //Cálculo de la constante: \Phi = \begin{bmatrix} \phi_{00} & \phi_{01} \\ \phi_{10} & \phi_{11} \end{bmatrix} \in \mathbb{R}^{2 \times 2}.
24
            Phi[0][0]=exp(-0.1*h)*(cos(0.995*h)+0.1*sin(0.995*h)/0.995);// \phi_{00}.
25
            Phi[0][1]=exp(-0.1*h)*sin(0.995*h)/0.995;// \phi_{01}.
26
            Phi[1][0]=-exp(-0.1*h)*sin(0.995*h)/0.995;// \phi_{10}.
27
            Phi[1][1]=exp(-0.1*h)*(cos(0.995*h)-0.1*sin(0.995*h)/0.995);// \phi_{11}.
28
```

```
Continúa código Arduino 9.4a: sketch cap9_sistema2dokC
   Arduino. Aplicaciones en Robótica y Mecatrónica.
   Capítulo 9 Control.
   Fernando Reyes Cortés y Jaime Cid Monjaraz.
   Alfaomega Grupo Editor: "Te acerca al conocimiento".
   Continuación del sketch cap9_sistema2dokC.ino
29
           tk=0;// tiempo discreto t_k inicia en cero segundos.
           u=0;//ley de control u(0)=0.
30
           //Condición inicial del vector de estados: \boldsymbol{x}(0) = \begin{bmatrix} x_{00}(0) \\ x_{10}(0) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.
31
           xk[0][0]=xka[0][0]=0;//x_{00}(0)=0, xka significa estado anterior x_{00}(t_{k-1}).
32
           xk[1][0]=xka[1][0]=0;//estado anterior <math>x_{10}(t_{k-1})=x_{10}(t_k).
33
           do{//protocolo de comunicación y enlace entre Arduino y MATLAB.
34
              Serial.print(33.0000,4);
35
              Serial.print(" ");
36
37
              Serial.print(h,4);
              Serial.print(" ");
              Serial.println(base_tiempo/h,4);
39
              if(Serial.available()>0){
40
                  fase=Serial.read();
41
                  if (fase=95)
42
                     bandera=0;
43
                     Serial.print(0);
44
                     Serial.print(" ");
45
                     Serial.print(0);
46
                     Serial.print(" ");
47
                     Serial.println(0);
48
49
                     break;
50
              delay(600);
52
53
           \}while(1);
54
```

Continúa código Arduino 9.4b: sketch cap9_sistema2dokC Arduino. Aplicaciones en Robótica y Mecatrónica. Capítulo 9 Control. Fernando Reyes Cortés y Jaime Cid Monjaraz. Alfaomega Grupo Editor: "Te acerca al conocimiento". Continuación del sketch cap9_sistema2dokC.ino 55 if(tk<base_tiempo){//tiempo de ejecución en la tarjeta Arduino. for(i=0; i<2; i++){//algoritmo discreto: $\boldsymbol{x}(t_k) = \Phi \boldsymbol{x}(t_{k-1}) + \Gamma u(t_{k-1})$. 56 for(j=0; j<2; j++){//algoritmo para calcular la versión recursiva de $x(t_k)$. 57 58 xk[i][0]=Phi[i][j]*xka[j][0]+xk[i][0];**59** $xk[i][0]=xk[i][0]+Gamma[i][0]*u;//complementa cálculo recursivo de <math>x(t_k)$. 60 61 //Posición $x_1(t_k)$ está representada por xk[0][0], velocidad $x_2(t_k) = xk[1][0]$. 62 xtilde=xd-xk[0][0];//error de posición: $\tilde{x}(t_k) = x_d - x_1(t_k)$. 63 64 //Ley de control: $u(t_k) = k_p \tanh(\tilde{x}(t_k)) - k_v \tanh(x_2(t_k))$. 65 u=kp*tanh(xtilde)-kv*tanh(xk[1][0]);//lev de control envia $_$ datosMatlab(tk,xk[0][0],xk[1][0]);//envía datos a graficar a Matlab. 66 tk=tk+h;//evolución de la variable tiempo: $t_k=kh$. 67 delay(h*1000);//retardo de un milisegundo. 68 xka[0][0]=xk[0][0];//estado actual resguarda en estados anteriores. 69 $xka[1][0]=xk[1][0];//estado anterior <math>x_{10}(t_{k-1})=x_{10}(t_k).$ 70 71 xk[0][0]=xk[1][0]=0;//se limpia registros para manipulación aritmética. **72**

La figura 9.5 muestra la respuesta $x_1(t)$ de simulación del sistema (9.12a) con la función ode45(...) y el algoritmo discreto (9.13a) que se ejecuta en la tarjeta Arduino.

73 $\}/\text{fin de la subrutina principal loop()}\{...\}, línea 19.$

Note que se han eliminado las oscilaciones y sobreimpulsos que se presentaron en la respuesta en lazo abierto (figura 9.3); esto se debe a las ventajas que presenta el efecto de retroalimentación y a la incorporación de un elemento de amortiguamiento usando una función saturada.

La respuesta retroalimentada es suave, sin sobreimpulsos, entrando en estado estacionario en 8 segundos, libre de oscilaciones y rizo.

La ejecución del sketch **cap9_sistema2dok**C con los scripts del modelo dinámico y control no lineal (**cap9_simusistema2doA** y **cap9_sistema2doCC**, respectivamente) es similar al descrito en la página 298.

```
A Código MATLAB 9.5 cap9_sistema2doCC.m
 Arduino. Aplicaciones en Robótica y Mecatrónica.
 Capítulo 9 Control.
 Fernando Reyes Cortés y Jaime Cid Monjaraz.
 Alfaomega Grupo Editor: "Te acerca al conocimiento".
  Archivo cap9_sistema2doCC.m
                                                     Versión de Matlab 2014a
1 function xp=cap9_sistema2doCC(t,x)
     kp=20; kv=20; xd=1;
3
     wn=1; rho=0.1;
     A=[0 1; -wn*wn -2*rho*wn];
4
     B = [0; 1];
5
     xtilde=xd-x(1,1);
6
     u=kp*tanh(xtilde)-kv*tanh(x(2,1));
7
     xp=A*x+B*u;
8
9 end
```

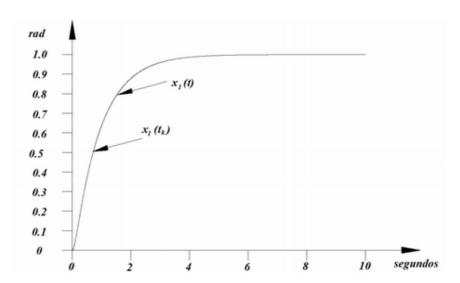


Figura 9.5 Posición discreta $x_1(t_k)$ Arduino vs posición $x_1(t)$ simulada en MATLAB.

♣ ♣ Ejemplo 9.3

Analizar la respuesta a un escalón de voltaje de 1 V para la función de transferencia (9.2) implementada a través de un amplificador operacional con componentes resistivos y capacitivos.

Solución

La función de transferencia (9.2) describe el comportamiento de un sistema lineal y se modela a través del factor de amortiguamiento ρ y de la frecuencia natural de oscilación w_n :

$$\frac{y(s)}{u(s)} = \frac{w_n^2}{s^2 + 2\rho w_n s + w_n^2}$$

el factor de amortiguamiento ρ describe el efecto disipativo (proceso térmico que convierte la energía eléctrica a energía térmica) para eliminar los sobreimpulsos y oscilaciones en la respuesta transitoria del sistema.

Esta función de transferencia (9.2) representa el modelo de un sistema físico, por ejemplo un circuito electrónico realizado con amplificadores operacionales con resistencias y capacitores, cuya estructura se puede expresar de la siguiente forma:

$$\frac{y(s)}{u(s)} = \frac{\left(\frac{R_4}{R_3} + 1\right) \frac{1}{R_1 R_2 C_1 C_2}}{s^2 + \left[\frac{1}{C_1} \left(\frac{1}{R_1} + \frac{1}{R_2}\right) - \frac{1}{R_2 C_2} \frac{R_4}{R_3}\right] s + \frac{1}{R_1 R_2 C_1 C_2}}$$
(9.30)

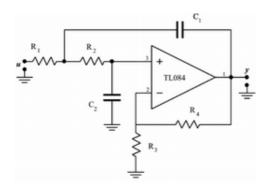


Figura 9.6 Sistema dinámico lineal de segundo orden.

La síntesis de la función de transferencia (9.30) es por medio del diagrama electrónico de la figura 9.6, por tal motivo se requiere de una tarjeta Arduino que incluya convertidor digital/analógico (además del convertidor analógico/digital). El modelo Arduino Due cubre estos requerimientos, la arquitectura electrónica contiene un microcontrolador ARM CortexM3 de 32 bits a 84 Mhz, 2 DAC's de 12 bits y 12 entradas analógicas/digital de 12 bits.

El rango de voltaje con el que trabaja Arduino Due es de 0 V a 3.3 V, por lo que un voltaje mayor a ese límite dañaría sus componentes

electrónicos. Sin embargo, los dos DAC's de la tarjeta presentan desventajas funcionales, por ejemplo no manejan voltajes negativos; además, en 0 V cada DAC proporciona una salida de offset alrededor de $522~\mathrm{mV}$, es decir, no pueden entregar 0 V. Esto es una limitante importante que se debe tomar en cuenta durante la parte experimental.

El voltaje de salida de 522 mV para u(t) = 0 V representa una señal indeseable, es un corrimiento (offset), el cual no es constante, depende de la temperatura y tiene fluctuaciones en el tiempo; no obstante, se puede suponer que ambos DAC's tienen estadísticamente el mismo offset. Por lo tanto, se empleará instrumentación electrónica para eliminar el offset y acondicionar u(t) para la adquisición de datos en la tarjeta Arduino Due; la idea principal se muestra en la figura 9.7.

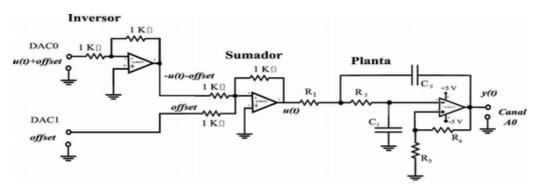


Figura 9.7 Acoplamiento de u(t) y cancelación de la señal de offset del DACO.

El convertidor DAC0 tiene la señal de offset más la señal del escalón unitario de un Volt u(t), se conecta a un amplificador inversor de ganancia unitaria, en la salida de este amplificador inversor se obtiene: -offset-u(t), este resultado se conecta a un sumador inversor, donde en una de sus entradas se tiene la señal de offset del DAC1; la salida del sumador tiene: -(offset-offset-u(t))=u(t), ya es una señal de entrada con las características adecuadas para aplicarse a la planta de estudio.

Los amplificadores operacionales que se utilizan están contenidos en el circuito integrado TL084CN y se emplean fuentes externas de alimentación ± 5 V, mientras que la tarjeta Arduino Due se alimenta del cable mini-USB; ambas tierras (referencias eléctricas) de la tarjeta Arduino y de los amplificadores operacionales se encuentran unidas.

La función de transferencia (9.30) adquiere un formato numérico con los siguientes valores de componentes: frecuencia natural de resonancia $w_n=1$ rad/seg y el factor de amortiguamiento $\rho=0.4$. Se han seleccionado $R_1=R_2=1$ M Ω , $R_3=1$ K Ω , $R_4=1.2$ K Ω y $C_1=C_2=1$ μ F:

$$\frac{y(s)}{u(s)} = \frac{\left(\frac{R_4}{R_3} + 1\right) \frac{1}{R_1 R_2 C_1 C_2}}{s^2 + \left[\frac{1}{C_1} \left(\frac{1}{R_1} + \frac{1}{R_2}\right) - \frac{1}{R_2 C_2} \frac{R_4}{R_3}\right] s + \frac{1}{R_1 R_2 C_1 C_2}}$$

$$= 2.2 \frac{1}{s^2 + 0.8s + 1} \tag{9.31}$$

los valores de estos componentes corresponden a la implementación práctica de la planta de segundo orden de la figura 9.7.

El cuadro de código Arduino 9.6 contiene el sketch $\operatorname{cap9_OPamLA}$ para adquirir la respuesta de la función de transferencia para una entrada escalón u(t) de magnitud de un Volt y de la etapa

de instrumentación electrónica implementada a través de operacionales. El periodo de muestreo ha sido seleccionado en 10 milisegundos; de la línea 1 a la 4 se encuentran declaradas las variables globales requeridas por el sketch. El tiempo de adquisición es de 20 segundos (dos mil puntos de registro).

La línea 5 contiene la función encargada de enviar información al ambiente de programación MATLAB. La subrutina de configuración setup() se ubica en la línea 12, donde se establece la velocidad de comunicación serial en 19200 Baudios y se configura la resolución en 12 bits para el convertidor analógico/digital y los DAC's, es decir, quedan operando con 4096 niveles cuantificables. La rutina de programación principal loop() inicia a partir de la línea 17.

El protocolo de comunicación serial que coordina el intercambio de información entre la tarjeta Arduino Due y el script cap9_Graficar_Con_Matlab en MATLAB (ver cuadro de código en MATLAB 9.7) está contenido entre las líneas 18 y 42. A partir de la línea 43 se encuentra el algoritmo de registro de la respuesta de la planta, la línea 44 muestra la conversión a voltaje de los niveles cuantificados del ADC, tomando en cuenta el voltaje máximo de 3300 mV y 12 bits de resolución (4096 pasos de cuantificación en la conversión analógica a digital).

Otra de las desventajas que presentan los DAC's es que no están debidamente calibrados, esto significa que la codificación del convertidor digital/analógico no coincide con la magnitud de voltaje de la señal de salida en la conversión. Por tal motivo, en la línea 49 se hace un sencillo ajuste para obtener una lectura lo más cercana al valor real de voltaje que debe entregar el DACO.

En la línea 51 se envía a **MATLAB** la respuesta de la planta (función de transferencia (9.30) implementada con un amplificador operacional). La evolución del tiempo desde cero hasta 20 segundos se lleva a cabo en la línea 52 con pausas de tiempo de 10 milisegundos (línea 53).

Es importante resaltar las siguientes acotaciones sobre la tarjeta Arduino Due:



Tomando en cuenta la selección de la tarjeta Arduino Due, la compilación del sketch cap9_OPamLA debe realizarse exclusivamente con la versión 1.57 (o posterior) del IDE Arduino, que puede ser descargada de: www.arduino.cc. De otra manera, con versiones anteriores no podrá ser compilado.



La versión 1.57 del IDE Arduino ya incluye los identificadores DAC0, DAC1 y todas las características de programación para la tarjeta Arduino Due.



En el menú **Herramientas** del IDE Arduino, seleccionar la tarjeta Arduino Due (programming port).



Es recomendable conectar la tarjeta Arduino Due a la computadora a través del minipuerto USB (programming port).



Los 12 ADC's, 2 DAC'2, así como los 54 puertos digitales trabajan con voltajes comprendidos entre 0 a $3.3~\rm V$, no exceda el límite máximo de $3.3~\rm V$ para no dañar los componentes de la tarjeta.



Mayor información sobre la instalación y uso de la tarjeta Arduino Due se encuentra disponible en el material Web del Capítulo 2 **Instalación y puesta a punto del sistema Arduino**.



Ejemplos con Arduino Due

En el sitio Web de este libro se han considerado una serie de sketchs con algoritmos de adquisición de datos y control utilizando el modelo de la tarjeta Arduino Due.

El modelo de tarjeta Arduino Due requiere la versión 1.57 o posterior del IDE que puede ser descargada de: www.arduino.cc

El cuadro de código MATLAB 9.7 describe el script cap9_Graficar_Con_Matlab, este programa permite enlazar la comunicación serie entre la tarjeta Arduino y MATLAB para graficar la información de las variables durante la ejecución del sketch en la tarjeta Arduino Due; contiene el mismo código de los ejemplos previos. Es importante verificar que el puerto serial (línea 2) sea el mismo que utiliza la tarjeta Arduino Due.

```
    Código Arduino 9.6: sketch cap9_OPamLA

  Arduino. Aplicaciones en Robótica y Mecatrónica.
  Capítulo 9 Control.
  Fernando Reyes Cortés y Jaime Cid Monjaraz.
  Alfaomega Grupo Editor: "Te acerca al conocimiento".
   Sketch cap9_OPamLA.ino
 1 float tk=0, y, u;
 2 float h=0.01;//periodo de muestreo de 10 milisegundos.
 3 int fase=0, bandera=1;
 {\tt 4}\,float base_tiempo=20;//tiempo de experimentación en 20 segundos.
 5 void envia_datosMatlab(float t, float var1, float var2){
       Serial.print(t);
       Serial.print(" ");
       Serial.print(var1);
       Serial.print(" ");
10
       Serial.println(var2);
11 }
12 void setup() {//rutina de configuración.
       Serial.begin(19200);//velocidad de comunicación serie en 19200 Baudios.
13
       analogWriteResolution(12);//resolución de la salida analógica a 12 bits (4096).
14
       analogReadResolution(12);//resolución del canal analógico a 12 bits (4096).
15
16 }
17 void loop(){//rutina principal de programación.
       if (bandera){//protocolo de comunicación serial entre Arduino y MATLAB.
         tk=0;
19
          do{//comandos de sincronía en la comunicación serial
20
             Serial.print(33.0000,4);
21
             Serial.print(" ");
22
             Serial.print(h,4);
23
             Serial.print(" ");
24
             Serial.println(base_tiempo/h,4);
25
             if(Serial.available()>0){
26
                fase=Serial.read();
27
```

Continúa código Arduino 9.6a: sketch cap9_OPamLA Arduino. Aplicaciones en Robótica y Mecatrónica. Capítulo 9 Control. Fernando Reyes Cortés y Jaime Cid Monjaraz.

Continuación del sketch cap9-OPamLA.ino

Alfaomega Grupo Editor: "Te acerca al conocimiento".

```
if (fase==95){//limpia registro serial}.
28
                   bandera=0;
29
                   Serial.print(0);
30
                   Serial.print(" ");
31
                   Serial.print(0);
32
                   Serial.print(" ");
33
                   Serial.println(0);
34
                   break;
35
36
             }
37
             analogWrite(DAC0, 0);//genera señal de offset en DAC0.
38
             analogWrite(DAC1, 0);//genera señal de offset en DAC1.
39
40
             delay(800);
          \}while(1);
41
       }
42
       if(tk<base_tiempo){//algoritmo de adquisición de datos
43
         y=3300.0*(analogRead(A0))/4096.0;
44
         if(tk==0)
45
             u = 0.0;
46
47
          else
             u=1000.0;
48
          analogWrite(DAC0, (u/1.4)*4096.0/3300.0);//correción del DAC0.
49
          analogWrite(DAC1, 0);//generar señal de offset.
50
          envia_datosMatlab(tk,y,u);//envía datos a graficar en MATLAB.
51
          tk=tk+h;//evolución de la variable tiempo: t_k = kh.
52
          delayMicroseconds(h*1000000.0);//retardos de 10 milisegundos.
53
54
55 }
```

```
📣 Código MATLAB 9.7 cap9_Graficar_Con_Matlab.m
   Arduino. Aplicaciones en Robótica y Mecatrónica.
   Capítulo 9 Control.
   Fernando Reyes Cortés y Jaime Cid Monjaraz.
   Alfaomega Grupo Editor: "Te acerca al conocimiento".
   Archivo cap9_Graficar_Con_Matlab.m
                                                            Versión de Matlab 2014a
 1 clc; clear all; close all; format short g
 2 canal_serie = serial('COM10', 'BaudRate', 19200, 'Terminator', 'CR/LF'); %crea objeto serie.
 3 fopen(canal_serie); %abrir puerto.
 4 xlabel('Segundos'); ylabel('Datos'); title('Control de procesos utilizando Arduino');
 5 grid on; hold on; fase=0;
 6 disp('Sincronizando enlace con tarjeta Arduino')
       fase=fscanf(canal_serie, '%f%f%f',[3,1]);%lee puerto serie.
       disp(['Comando=', num2str(fase(1,1)), 'Periodo de muestreo=',...
       num2str(fase(2,1)), \ '\ N\'umero\ de\ puntos\ a\ graficar='\ ,\ num2str(fase(3,1))]);
       if (fase(1,1)==33.0000)
11
           disp('Inicia adquisición de datos con la tarjeta Arduino')
12
           break:
13
       end
14
15 end
16 fwrite(canal_serie, 95, 'int');
17 prop = line(nan,nan, 'Color', 'b', 'LineWidth',1);
18 fscanf(canal_serie, '%f%f%f',[3,1]);%basura en el puerto serial.
19 disp('Graficando variables de Arduino...');
20 for i=1:(fase(3,1))
       datos = fscanf(canal_serie, '%f%f%f',[3,1]);%leer el puerto serie.
21
       tiempo(i,1) = datos(1,1); var1(i,1) = datos(2,1); var2(i,1) = datos(3,1);
22
       set(prop, 'YData', var1(1:i,1), 'XData', tiempo(1:i,1));
23
24
       drawnow;
25 end
26 fclose(canal_serie); %cierra objeto serial.
27 delete(canal_serie); %libera memoria.
```

28 clear canal_serie; % .

La figura 9.8 muestra la respuesta de la función de transferencia (9.31) a una entrada escalón de 1000 mV, observe que la fase transitoria contiene un sobreimpulso, seguido por oscilaciones decrecientes que se desvanecen en el tiempo hasta alcanzar el estado estacionario. Este tipo de comportamiento en la fase transitoria es síntoma inequívoco de un efecto de amortiguamiento con factor $\rho < 1$.

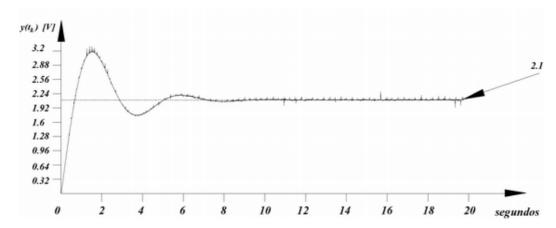


Figura 9.8 Respuesta a un escalón de 1 V en lazo abierto de la planta física.

La ejecución del sketch **cap9_OPamLA** con el script **cap9_Graficar_Con_Matlab** es similar al descrito en la página 298. Es importante reiterar que la versión 1.5.7 o posterior del IDE de Arduino es el software adecuado para programar la tarjeta Due.



9.2.2 Aspectos técnicos a considerar en las tarjetas Arduino

El sistema Arduino tiene una gran cantidad de aplicaciones en ingeniería y ciencias exactas, particularmente en la automatización de procesos. Sin embargo, como ya se puso en evidencia en el ejemplo 9.3, los detalles técnicos para acoplar la señal a la etapa de adquisición de datos de las tarjetas Arduino no son triviales, así como para enviar la señal de control a la planta física, en varias ocasiones se requiere de instrumentación electrónica, debido a que los ADC's y los DAC's no manejan voltajes negativos. Esto no es un obstáculo, se requiere diseñar una interface electrónica adecuada que de manera ex profeso cubra todos los requerimientos y características del proceso físico.

Por ejemplo, para controlar la señal de la planta (9.31) a una referencia deseada por medio de esquemas de control, se requiere enviar a dicha planta voltaje positivo y negativo; considere el caso donde se tiene un esquema de control retroalimentado, y la ley de control determina aplicar voltaje negativo para reducir la magnitud del error, es decir, descargar a los capacitores durante un número

determinado de periodos de muestreo y así conducir al estado de la planta hacia la referencia.

Con los circuitos ADC's y DAC's de las tarjetas Arduino no sería posible, ya que no manejan voltajes negativos, por lo tanto se requiere diseñar una interface que incluya amplificadores operacionales, circuitos comparadores, atenuadores, limitadores, sumadores de voltaje, entre otros componentes más, para acondicionar la respuesta y de la ley de control de la planta en un rango específico, por ejemplo: de 0 a 1.65 maneja voltajes negativos y de 1.65 a 3.3 representando voltajes positivos, siendo cero volts igual a 1.65 V.

En otras aplicaciones se requiere el acoplamiento de instrumentación especializada como es el caso de los FPGA's para leer la información del encoder del servomotor y proporcionar el desplazamiento articular del eje de giro del rotor, así como el empleo de circuitos de potencia para entregar la energía con la corriente de consumo adecuada, este es el caso de sistemas empleados en robótica y mecatrónica.



Algoritmos de control

En el sitio Web de esta obra se describe la instrumentación electrónica para controlar la salida de la planta (9.30) por medio de retroalimentación de estados utilizando el modelo Arduino Due. Se presentan diversas estrategias de control no lineales, diseño con el enfoque de la teoría de estabilidad de Lyapunov e implementación práctica.



Control de un péndulo

Otro de los sistemas de las áreas de robótica y mecatrónica que requiere de una interface adecuada es un péndulo, el cual utiliza un servomotor con encoder para entregar la información del desplazamiento articular. Es clave el diseño de la interface electrónica con convertidores digital/analógico para enviar comandos a la etapa de potencia y lograr que la posición del extremo final del péndulo converja a la referencia deseada. Estos aspectos se describen el el sitio Web de la presente obra.



9.3 Control de temperatura

Dentro de las principales aplicaciones industriales se encuentran los procesos térmicos, que van desde una simple elaboración de alimento, fundición y moldeo de metales, recocido de materiales, procesado de pintura automotriz, tratamiento de cerámica hasta la fabricación de sensores y dispositivos semiconductores. La forma de manipular la temperatura es a través de un horno el cual es un sistema que tiene la capacidad de incrementar la temperatura, aislarla del medio ambiente y mantenerla dentro de un compartimiento para procesar las propiedades de diversos tipos de materiales. La energía térmica es la fuente de funcionamiento de los hornos y generalmente se obtiene por la combustión de diferentes materiales o por medio de electricidad. La temperatura que puede manejar un horno industrial puede ir desde 100 °C hasta niveles superiores a 3000 °C; la parte clave de un horno industrial es el control de temperatura.

La figura 9.9 muestra el diagrama a bloques básico de un horno industrial el cual eleva exponencialmente la temperatura por medio de una resistencia o calefactor eléctrico, incluye un control de temperatura, etapa de potencia y sensor de temperatura que en algunos casos requiere una etapa de instrumentación electrónica para acondicionar la señal a un sistema digital.

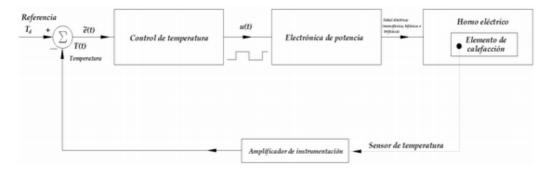


Figura 9.9 Diagrama a bloques de un horno industrial eléctrico.

La temperatura deseada T_d es la referencia (set-point) donde se desea que el horno esté controlado, T(t) es la temperatura del interior del horno cuya información es proporcionada por un sensor de temperatura generando una señal voltaje proporcional a T(t); ejemplos de sensores de temperatura: termopar, termistor, termorresistencia y una diversidad de dispositivos electrónicos. El error de temperatura $\tilde{e}(t)$ se define como la diferencia entre la temperatura deseada y la temperatura real del horno, es decir: $\tilde{e}(t) = T_d - T(t)$, esta señal es procesada por un algoritmo de control que garantiza la convergencia asintótica hacia cero (punto de equilibrio), es decir: $\lim_{t\to\infty} \tilde{e}(t) \to 0$, en otras palabras, la temperatura interna del horno alcanza la referencia deseada: $T(t) \to T_d$. La salida del algoritmo de control u(t) se convierte a una señal por modulación de ancho de pulso (PWM: pulse width modulation) para acoplarse a la etapa de potencia a través de un optoacoplador

(preferentemente por cruce de cero, por ejemplo MOC3041), el cual acciona la compuerta de un elemento electrónico de potencia como puede ser tiristor, triac o SCR; de esta forma se controla el suministro de energía eléctrica (120 VAC, bifásica 220 VAC o trifásica 440 VAC) del elemento de calefacción del horno para incrementar/decrementar la temperatura del interior del horno.



9.3.1 Control de temperatura PID

Dentro de los esquemas de control más utilizados en procesos térmicos se encuentra el proporcionalintegral-derivativo (PID) cuya estructura matemática se muestra a continuación:

$$u(t) = k_p \tilde{e}(t) + k_i \int_0^t \tilde{e}(\sigma) d\sigma - k_v \dot{T}(t)$$
(9.32)

donde $k_p, k_i, k_v \in \mathbb{R}_+$ representan las ganancias proporcional, integral y derivativa, respectivamente; $\tilde{e}(t) \in \mathbb{R}$ es el error de temperatura, $\dot{T}(t) \in \mathbb{R}$ es la derivada con respecto al tiempo de la temperatura, representando su variación temporal o velocidad de movimiento del proceso térmico; $u(t) \in \mathbb{R}$ es la salida o respuesta del control PID, cuya señal se convierte a PWM para conectarse a la etapa de potencia y suministrar energía a la resistencia térmica del horno, $t \in \mathbb{R}_+$ es la evolución del tiempo $(t \geq 0)$.

Observe que la derivada del error de temperatura $\dot{\tilde{e}}(t) = \frac{d}{dt}T_d - \frac{d}{dt}T(t) = -\frac{d}{dt}T(t) = -\dot{T}(t)$, debido a que la referencia T_d es una constante. En la ecuación (9.32) el primer término $k_p\tilde{e}(t)$ representa el control proporcional (también conocido como regulador), es el único elemento de control, ya que los términos $k_i \int_0^t \tilde{e}(\sigma)d\sigma - k_v\dot{T}(t)$ deben ser interpretados como acciones de control integral y derivativa, respectivamente. Es decir, no son esquemas de control más bien, son elementos necesarios que corrigen o perfeccionan las características analíticas y cualitativas del control proporcional $k_p\tilde{e}(t)$. La acción de control integral ayuda a reducir el error en estado estacionario que mantiene el control proporcional; sin embargo, también tiene efecto en la etapa transitoria, aquí es donde se desea que modifique la respuesta lo menor posible, por este motivo la ganancia de control integral k_i debe ser pequeña. La acción de control derivativa funciona como un amortiguador o elemento disipativo que elimina sobreimpulsos y oscilaciones en la fase transitoria, en estado estacionario la temperatura del horno T(t) es constante, idealmente alcanza la temperatura deseada T_d y se mantiene fija, por lo que su derivada es cero, de ahí que la ganancia derivativa k_d debe ser seleccionada adecuadamente.

Existen varias formas y técnicas de sintonizar las ganancias del control PID, la gran mayoría son de naturaleza empírica, es decir, dependen de la experiencia. Una de ellas se define por la selección de una banda proporcional en la respuesta del horno; el control proporcional $k_p\tilde{e}(t)$ mantiene una banda de respuesta donde se ajustan los límites inferior y superior de la banda. La figura 9.10 muestra este concepto.

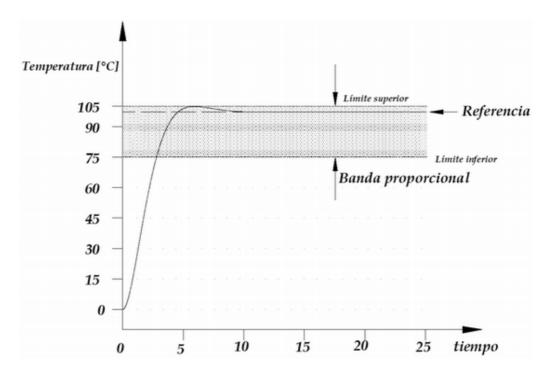


Figura 9.10 Banda proporcional del proceso térmico.

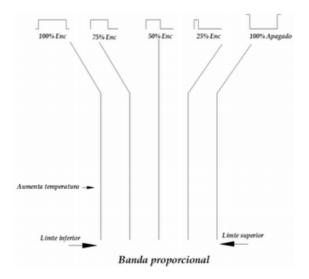


Figura 9.11 Porcentaje de encendido/apagado de la banda proporcional.

Cuando el horno se enciende, inicia el incremento de la temperatura; sin embargo, T(t) es menor al límite inferior de la banda proporcional, mientras la temperatura se mantenga por debajo de ese límite representa el 100 % de calefacción, una vez que la temperatura del horno T(t) entra en la banda, el tiempo de calefacción (encendido) se decrementa proporcionalmente, la acción derivativa introduce un amortiguador o freno en la subida de T(t), por lo que gradualmente se incrementa el tiempo de apagado, cuando la temperatura T(t) del proceso alcanza el centro de la banda proporcional, la calefacción estará al 50 %, el límite superior de la banda representa el

 $100\,\%$ de apagado. La figura 9.11 muestra el porcentaje de encendido/apagado, el cual es directamente proporcional al suministro de energía eléctrica de un horno industrial.



9.3.2 Regla de sintonía del control de temperatura PID

La regla de sintonía es un procedimiento que permite obtener el valor numérico de las ganancias proporcional k_p , integral k_i y derivativa k_v , tal que se tenga una respuesta adecuada para el horno. Este procedimiento depende de las características dinámicas del proceso, por lo que esta regla de sintonía para el control PID aplica sólo para hornos industriales y no para otros tipos de plantas como el sistema lineal de segundo orden analizado en la sección 9.2 o sistemas mecatrónicos como un péndulo o robot. Cada sistema o planta física determina un proceso de sintonía con base en sus características dinámicas y generalmente es diferente para cada proceso.

La banda proporcional representa la zona en que la señal del error de temperatura $\tilde{e}(t)$ se desplaza para generar conmutación por modulación de ancho de pulso en la línea de potencia. La banda proporcional B_p se define como la diferencia entre el límite superior y el límite inferior de temperatura; dichos límites son propuestos por el usuario. El límite superior delimita el máximo sobreimpulso que pueda tener T(t) (por ejemplo, 1.1 T_d), mientras que el límite inferior representa un porcentaje de la temperatura de referencia (por ejemplo, el 70 % de T_d). Por lo tanto, la ganancia proporcional k_p puede ser definida en términos de banda proporcional B_p de la siguiente forma:

$$k_p = \frac{100}{B_p} = \frac{100}{\text{límite superior - límite inferior}}$$
 (9.33)

Sea α el límite superior de la banda proporcional B_p y β representa el límite inferior, entonces se obtiene:

$$k_p = \frac{100\%}{B_p} = \frac{100\%}{\alpha - \beta}$$
 (9.34)

Las ganancias integral k_i y derivativa k_d se calculan en función de la ganancia proporcional k_p y en términos de otros parámetros denominados tiempo integral $t_i \in \mathbb{R}_+$ y tiempo derivativo $t_d \in \mathbb{R}_+$:

$$k_i = \frac{k_p}{t_i} \tag{9.35a}$$

$$k_d = t_d k_p (9.35b)$$

cuando el parámetro tiempo integral t_i es grande (es decir, la proporción $\frac{k_p}{t_i}$ es pequeña), la acción de control integral responde más lento, causando menos efecto en la fase transitoria, su principal aportación se ubica en el estado transitorio disminuyendo el error de temperatura. El error $\tilde{e}(t)$ es integrado con la acción de control integral, es decir es la acumulación de áreas bajo la curva del error de temperatura $\tilde{e}(t)$ en cada instante de tiempo (en cada período de muestreo), por lo que la magnitud que puede alcanzar la acción de control integral $\int_0^t \tilde{e}(\sigma) d\sigma$ puede ser grande, debido a esto se requiere que la ganancia de control integral $k_i = \frac{k_p}{t_i}$ deba ser muy pequeña, lo anterior se logra seleccionando un tiempo integral grande. Por otro lado, el tiempo derivativo t_d afecta proporcionalmente a la acción de control derivativa, generando un freno mecánico o amortiguador grande si t_d es grande, la respuesta del horno será sobreamortiguada, evitando sobreimpulsos

y oscilaciones; cuando t_d es muy pequeña, la acción de control derivativa no puede detener el incremento de temperatura durante el estado transitorio y por lo tanto, tendrá un comportamiento abrupto con picos, sobreimpulsos y oscilaciones.

Tomando en cuenta la regla de sintonía propuesta para las ganancias del control PID, u(t) adquiere la siguiente forma:

$$u(t) = k_p \tilde{e}(t) + \frac{100 \%}{t_i (\alpha - \beta)} \int_0^t \tilde{e}(\sigma) d\sigma - t_d \frac{100 \%}{\alpha - \beta} \dot{T}(t)$$

$$= k_p \left[\tilde{e}(t) + \frac{1}{t_i} \int_0^t \tilde{e}(\sigma) d\sigma - t_d \dot{T}(t) \right]$$
(9.36)

Suponga que el horno industrial tiene condiciones iniciales cero: $[T(0), \dot{T}(0)]^T = [0 \, ^{\circ}\text{C}, \, 0 \, ^{\circ}\text{C/segundo}]^T$, entonces observe que el error de temperatura en el instante cero satisface: $\tilde{e}(0) = T_d - T(0) = T_d$; por lo tanto, la ley de control PID en t = 0 es: $u(0) = \frac{100 \, \%}{\alpha - \beta} \tilde{e}(0) = \frac{100 \, \%}{\alpha - \beta} T_d$, este valor representa el máximo de u(t); en función de esto, la señal u(t) del control PID será normalizada al intervalo [0, 100] % para generar la secuencia correspondiente de apagado/encendido por medio de una señal modulada por ancho de pulso (PWM). Note que el error de temperatura $\tilde{e}(t)$ irá decreciendo exponencialmente hacia cero conforme se desplaza en la banda proporcional B_p .



9.3.3 Implementación práctica del control PID

La forma práctica de implementar el algoritmo de control PID u(t) (9.36) en las tarjetas Arduino es por medio de la versión discreta $u(t_k)$, donde t_k es el tiempo discreto, el cual se define como múltiplos del periodo de muestreo $h \in \mathbb{R}_+$, es decir $t_k = kh$, donde $k \in \mathbb{Z}_+$.

El esquema discreto del control PID (9.36) se obtiene como:

$$u(t_k) = k_p \left[\tilde{e}(t_k) + \frac{1}{t_i} \operatorname{Int}(t_k) - t_d \operatorname{euler}(t_k) \right]$$
(9.37a)

$$Int(t_k) = Int(t_{k-1}) + h\tilde{e}(t_k)$$
(9.37b)

$$euler(t_k) = \frac{T(t_k) - T(t_{k-1})}{h}$$
(9.37c)

donde $\operatorname{Int}(t_k)$ es la integral discreta usando el método de integración trapezoidal y euler (t_k) es la aproximación (derivada discreta) de $\dot{T}(t)$, obtenida por diferenciación numérica. Para este caso, los métodos numéricos para calcular la integral $\int_0^t \tilde{e}(\sigma)d\sigma$ y la derivada de la temperatura $\dot{T}(t)$ también se conocen como métodos de integración y diferenciación de Euler.

♣ ♣ ♣ Ejemplo 9.4

Controlar la temperatura en 250 °C de un pequeño horno eléctrico para fundido de piezas de plástico usando el esquema de control discreto PID (9.36).

Solución

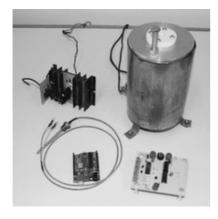


Figura 9.12 Horno eléctrico.

Algunas aplicaciones industriales requieren del fundido de material plástico para reciclar y crear nuevos productos; esta es la manera de reutilizar material de desecho, y según el tipo elementos de elementos con que se fabricó el plástico la temperatura para fundición puede oscilar entre 200 °C a 450 °C. El horno de temperatura que se ha seleccionado es de la compañía General Electric, es un horno pequeño con resistencia de tungsteno que le permite alcanzar hasta 300 °C al interior del horno y sirve para derretir objetos de plástico; este horno se conecta a la línea de 120 VAC y puede consumir hasta 15 A; no incluye sensor ni control de temperatura. La figura 9.12 muestra el horno

eléctrico, el sensor de temperatura seleccionado (termopar Chromel-Alumel, tipo JK con un rango hasta 1350 °C) y el interruptor electrónico de potencia es un triac (MAC15A).

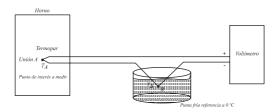


Figura 9.13 Puntas caliente T_A y

fría T_B del termopar.

El funcionamiento del termopar se basa en el principio de que dos metales al unirse producen efecto termoeléctrico (descubierto en 1821 por Thomas Johann Seebeck). El termopar es la unión de dos metales diferentes, para formar la unión A (punta caliente, T_A) que se encuentra en contacto con la temperatura que se desea medir (ver figura 9.13), los otros extremos del termopar se encuentran abiertos y son los que

proporcionan la diferencia de potencial; a estos dos extremos se les denomina unión fría debido a que se encuentran a temperatura ambiente y generalmente esta temperatura es diferente a la que se desea medir.

Para realizar una medición correcta se debe utilizar la compensación de unión fría donde la unión B (T_B) formada por una terminal del termopar y una punta del voltímetro se colocan a una temperatura con referencia conocida, por ejemplo a 0 °C; la otra terminal del termopar va directa a la otra punta del instrumento de medición.

El voltaje en los extremos del termopar está dado por la siguiente expresión:

$$V_{\text{Termopar}} = \beta_{\text{Termopar}} (T_A - T_B)$$
 (9.38)

donde β_{Termopar} es el coeficiente de Seebeck, generalmente en unidades de mV/°C, para el termopar Chromel-Alumel tipo JK, $\beta_{\text{Termopar}} = 40 \ \mu\text{V}/^{\circ}\text{C}$; T_A es la temperatura a medir (en la unión A) y T_B es la temperatura ambiente (unión B).

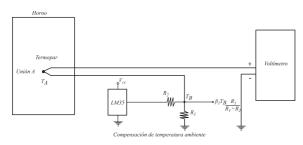


Figura 9.14 Compensación de temperatura ambiente.

Mantener la unión fría en una referencia fija sería muy complicado, por ejemplo artificialmente con hielos en 0 °C, en lugar de esto se emplea un dispositivo compensador de temperatura ambiente. La figura 9.14 muestra una forma posible de compensar la temperatura ambiente a través del dispositivo LM35, el cual tiene un coeficiente de Seebeck $\beta_{\rm LM35}$ de 10 mV/°C.

Tomando en cuenta la incorporación del compensador de unión fría por el dispositivo LM35 (en la unión B, T_B) la temperatura de medición está dada por la siguiente relación:

$$V_{\text{Medición}} = \beta_{\text{Termopar}} (T_A - T_B) + V_{\text{Compensador}}$$

 $= \beta_{\text{Termopar}} T_A - \beta_{\text{Termopar}} T_B + \beta_{\text{LM35}} T_B \frac{R_1}{R_1 + R_2}$ (9.39)

Para anular el efecto de temperatura ambiente en el termopar, de la ecuación (9.39) las resistencias R_1 y R_2 del divisor de voltaje deben ser seleccionadas tal que:

$$\beta_{\text{Termopar}} T_B = \beta_{\text{LM35}} T_B \frac{R_1}{R_1 + R_2} \tag{9.40}$$

Es decir, se debe satisfacer:

$$V_{\text{Medición}} = \beta_{\text{Termopar}} T_A - \beta_{\text{Termopar}} T_B + \beta_{\text{LM35}} T_B R_1 + R_2$$
 (9.41)

El termopar entrega una diferencia de potencial proporcional a la diferencia de temperatura entre la punta caliente y la punta fría (esta última se encuentra a temperatura ambiente).

La relación matemática que describe la temperatura del punto de interés (unión A, T_A) con la temperatura del otro extremo del termopar, es decir con la punta fría (unión B, T_B) con compensación de temperatura ambiente está dada por la siguiente relación:

$$V_{\text{Medición}} = \beta_{\text{Termopar}} T_A$$
 (9.42)

En la figura 9.14, en el punto B el divisor de voltaje $\frac{R_1}{R_1+R_2}$ tiene la finalidad de hacer compatible la respuesta del sensor LM35 con la lectura del termopar, seleccionado los siguientes valores de resistencias: $R_1 = 4 \text{ K}\Omega \text{ y } R_2 = 996000\Omega$, entonces la lectura del dispositivo compensador LM35 se convierte en:

$$\beta_{\text{LM35}} \frac{R_1}{R_1 + R_2} = 10 \text{ mV/}^{\circ} \text{C} \frac{4\text{K}\Omega}{4\text{K}\Omega + 0.996\text{M}\Omega} = 40 \mu\text{V/}^{\circ} \text{C}$$

El valor de las resistencias R_1 y R_2 se pueden ajustar con valores comerciales disponibles de la siguiente forma: $R_1 = 3.9 \text{ K}\Omega + 100 \Omega = 4 \text{ K}\Omega$ y para $R_2 = 820 \text{ K}\Omega + 150 \text{ K}\Omega + 22 \text{ K}\Omega + 3.9 \text{ K}\Omega + 100 \Omega = 996000 \Omega$.



Figura 9.15 Compensador de temperatura ambiente.

LM35 (National Semiconductor) es un dispositivo electrónico muy económico que permite sensar temperatura con voltaje de salida directamente proporcional a la medición de temperatura en °C, proporcionando 10 mV/°C. Tiene una exactitud de $\pm\frac{1}{4}$ °C a temperatura ambiente y $\pm\frac{3}{4}$ en su rango completo que va desde -55 °C a 155 °C. La fuente de alimentación puede ir desde 4 V a 30 V, tiene una corriente de pérdida menor a 60 μ A. La figura 9.15 muestra el encapsulado LM35D.

Los termopares tienen una respuesta lineal dentro de un amplio rango de temperatura, su construcción proporciona alta resistencia mecánica y fácil manejo. Sin embargo, este tipo de sensores de temperatura requieren de un circuito de acoplamiento antes de llegar al sistema digital Arduino.

Lo anterior se debe a que su señal es de baja magnitud, por ejemplo 40 $\mu V/^{\circ}C$, hay que tomar en cuenta que las tarjetas Arduino tienen convertidores analógico digital con una resolución de 10 bits, para una referencia de 5 V, la mínima lectura que puede discernir el sistema de adquisición es: $\frac{5\ V}{2^{10}} = \frac{5000\ mV}{1024} = 4.882\ mV.$ Con base en esta característica se recomienda obtener una señal de 5 mV/°C, por lo tanto se requiere de un amplificador de instrumentación con ganancia de 125.

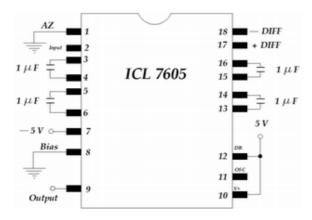


Figura 9.16 Amplificador de instrumentación.

La figura 9.16 muestra el amplificador de instrumentación seleccionado (ICL7605, INTERSIL), el cual contiene entrada diferencial y un subsistema de regulación automática de cero (CAZ) para eliminar la señal de offset que generalmente se presenta en la mayoría de los amplificadores operacionales. Este circuito opera en bajas frecuencias dentro del ancho de banda de DC a 10 Hz, por lo cual lo hace adecuado en aplicaciones de procesos térmicos que emplean sensores de temperatura con baja señal.

La figura 9.17 presenta la interface electrónica que se emplea en el control de temperatura para el horno eléctrico. La señal del termopar se encuentra acoplada a la tarjeta Arduino por medio del amplificador de instrumentación electrónica (ICL7605), en la entrada diferencial (DIFF-) se conecta la polaridad negativa del termopar, mientras que el hilo positivo del sensor de temperatura (unión T_B) se encuentra conectada en el divisor de voltaje del compesador de temperatura ambiente LM35 y a la entrada DIFF+.

La ganancia del amplificador de instrumentación es de 125, establecida por las resistencias: $\frac{R_3+R_4}{R_3}=125$, donde se ha seleccionado $R_3=1~\mathrm{K}\Omega~\mathrm{y}~R_4=24~\mathrm{K}\Omega$. El valor de R_4 se obtiene con los siguientes valores comerciales: 124 K Ω =120 K Ω +3.9 K Ω +100 Ω .

La salida del amplificador operacional se conecta a un filtro pasabajas con frecuencia de corte $f=\frac{1}{2\pi R_f C_f}=\frac{1}{2\pi 100~\mathrm{K}\Omega~\mathrm{1~\mu F}}=1.6~\mathrm{Hz}$, la señal filtrada va directamente al canal analógico A0 de la tarjeta Arduino.

El suministro de energía (120 VAC) del horno eléctrico se realiza por medio de un dispositivo electrónico de potencia denominado triac (MAC15A, On Semiconductor), cuya compuerta es disparada por un optoacoplador por cruce de cero (MOC 3041, Motorola), permite el paso de corriente durante ciclos completos e inhibiendo la alimentación por ciclos completos; cruce por cero evita la generación de armónicas en la línea de alimentación, que ocurren en los circuitos llamados disparos por ángulo fase, causando distorsión y bajo factor de potencia, de esta forma no trabaja en la zona de histérisis de la resistencia eléctrica y por lo tanto, se reducen considerablemente las vibraciones mecánicas del calefactor. La relación entre los ciclos de paso de corriente y los inhibidos dan el control de potencia deseado.

El optoacoplador MOC3041 se dispara por medio del puerto digital pin 4, proporcionando una señal de modulación de ancho de pulso (PWM).

Como una forma de protección al puerto digital de la tarjeta Arduino, el optoacoplador se acciona por medio de un circuito inversor de colector abierto 74LS06 de Texas Instruments, se emplea una resistencia de pull-up de 1 K Ω ; también se añade un indicador LED (manejado por una compuerta inversor) para visualizar la activación del triac.

En el cuadro de código Arduino 9.8 se encuentra el sketch **cap9_PIDtemperatura** con el algoritmo de control PID en tiempo discreto para procesos de temperatura. De la línea 1 a la 7 se encuentran definidas todas las variables globales requeridas en la implementación de $u_{\text{PID}}(t_k)$. A partir de la línea 8 se encuentra la rutina **envia_datosMatlab(...)** para enviar información y graficar en **MATLAB**, la cual se explicó en ejemplos anteriores.

La función de configuración setup() establece la velocidad de comunicación serial en 19200 Baudios y define el puerto digital 4 como salida, con este puerto se activará la señal PWM como se ilustra en la subrutina Convierte_PID_pulsos_PWM(float u_PID)(...), la cual inicia a partir de la línea 19. La frecuencia del pulso PWM es de 100 Hz o 10 milisegundos, la forma de



Ejemplos con Intel Galileo

El control de temperatura PID y otro tipo de esquemas de control, métodos numéricos y algoritmos de procesamiento de señales son desarrollados para la tarjeta Arduino Intel Galileo.

En el sitio Web de la presenta obra se encuentra disponibles ejemplos utilizando este tipo de modelo, el cual requiere Arduino IDE para Intel Galileo. Se presenta el método de instalación y la puesta a punto de este modelo.

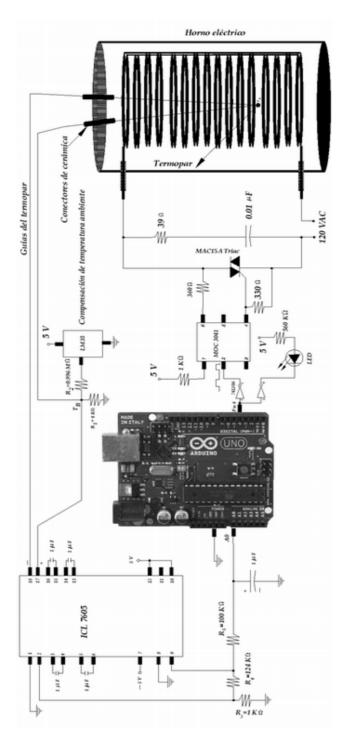


Figura 9.17 Interface electrónica del horno eléctrico para fundición de plástico.

modular la parte activa del pulso (cuando el puerto digital se encuentra en estado alto o 5 V) es normalizando la ley de control PID $u_{\text{PID}}(t_k)$, con respecto al periodo de muestreo h y del valor máximo (**cota_max**) es: $u_{\text{PID}}(0) = k_p \tilde{e}(0) = k_p T_d$.

En otras palabras, el tiempo que permanecerá encendido el horno será la parte proporcional de $\operatorname{ancho_pulso}=h\frac{u_{\mathrm{PID}}(t_k)}{\cot a_{\mathrm{max}}}$. El intervalo de normalización del algoritmo PID está restringido a: [0, cota_max]; observe que se multiplica por el factor 1000000 con la finalidad de acoplar adecuadamente el argumento de entrada (el cual está en microsegundos) de la función $\operatorname{delayMicroseconds}(\ldots)$, tomando en cuenta en la conversión que 1 mseg = 1000 μ seg. La parte proporcional del pulso PWM que estará en bajo (0 V), es decir apagado el horno se determina por: 1000000.0 $h\left(1-\frac{u_{\mathrm{PID}}(t_k)}{\cot a_{\mathrm{max}}}\right)$.

La subrutina de programación principal **loop()** se ubica en la línea 29 y la ejecución del algoritmo PID $u_{\text{PID}}(t_k)$ se realiza desde la línea 57, la adquisición de datos del horno (lectura de la temperatura) se lleva a cabo en la línea 58; se utiliza la ganancia del amplificador (125) para convertir a grados la lectura.

Note que en las líneas 60 y 62 se calculan la integral y derivada del error de temperatura por el método trapezoidal y de Euler, respectivamente. Sobre la línea 64 se lleva a cabo la implementación discreta del algoritmo PID, ecuación (9.37a).

Las líneas 66 y 69 tienen las funciones de envío de información para **MATLAB** y normalización y conversión de la ley de control PID a pulsos PWM, respectivamente.

Por otro lado, es muy importante aclarar algunos detalles técnicos en el estilo de programación referente a métodos numéricos de la integral discreta; la forma trapezoidal de la integral del error de temperatura es: $Int_k = Int_{k-1} + h\tilde{e}(t_k)$; sin embargo, como puede observar en la línea 60 se utiliza el código: Intk=Intk+h*error; note que se emplea la misma variable Intk en ambos lados de la igualdad, en la parte derecha del signo igual la variable Intk se interpreta de la siguiente manera: antes de realizar las operaciones aritméticas esta variable contiene el resultado de la iteración anterior, es decir, adquiere el papel de Int_{k-1} , al momento de asignar el resultado completo, entonces registra el cálculo de la iteración k-ésima en el lado derecho del signo igual.

Este es el mismo caso para la evolución en el tiempo discreto t_k , es decir $t_k = t_{k-1} + h$, observe que en la línea 67 utiliza el código $\mathbf{tk} = \mathbf{tk} + \mathbf{h}$. No obstante, es un caso diferente al de la línea 62 para estimar la derivada de la temperatura por diferenciación numérica, de ahí que se requiere actualizar la temperatura en el estado anterior $t(t_{k-1})$ como se indica en la línea 65.

```
€ Código Arduino 9.8: sketch cap9_PIDtemperatura
  Arduino. Aplicaciones en Robótica y Mecatrónica.
  Capítulo 9 Control.
  Fernando Reyes Cortés y Jaime Cid Monjaraz.
   Alfaomega Grupo Editor: "Te acerca al conocimiento".
   Sketch cap9_PIDtemperatura.ino
 1 float tk; //variable temporal discreta t_k.
 2 float h=0.01;//periodo de muestreo 10 milisegundos.
 3 int fase=0, bandera=1, pin4=4;
 4 float Temperatura, Temperatura, Td=250.0;//referencia T_d = 250 \, ^{\circ}\text{C}.
 5 float lim_inf, lim_sup, u_PID, cota_max;
 6 float error, Bp, kp, ti=300, td=0.008, Intk, Intka, euler;
 7 float base_tiempo=900;// tiempo experimental 15 minutos.
 8 void envia_datosMatlab(float t, float var1, float var2){//graficar en MATLAB.
       Serial.print(t);
 9
       Serial.print(" ");
10
11
       Serial.print(var1);
12
       Serial.print(" ");
       Serial.println(var2);
13
14 }
15 void setup() {//Rutina de configuración.
       Serial.begin(19200);//Comunicación serie en 19200 Baudios.
16
17
       pinMode(pin4, OUTPUT);//puerto digital 4 \rightarrow salida.
18 }
19 void Convierte_PID_pulsos_PWM(float u_PID){//PWM para encender/apagar el horno.
       float ancho_pulso;
20
21
       if(u\_PID <= cota\_max \&\& u\_PID >= 0) \ ancho\_pulso = 1000000.0*h*u\_PID/cota\_max;
       if(u_PID>cota_max) ancho_pulso=1000000.0*h;
22
       if(u_PID<0) ancho_pulso=0;
23
       digitalWrite(pin4, 1);//pulso en alto, enciende horno.
24
       delayMicroseconds(ancho_pulso);//parte proporcional: 10 \musegundos = 10 milisegundos.
25
       digitalWrite(pin4, 0);//pulso en bajo, apaga horno.
26
       delayMicroseconds(1000000.0*h-ancho_pulso);//1000000.0 h \left(1 - \frac{u_{\text{PID}}}{\cot a_{\text{max}}}\right).
27
28 }
```

```
Continúa código Arduino 9.8a: sketch cap9_PIDtemperatura
  Arduino. Aplicaciones en Robótica y Mecatrónica.
  Capítulo 9 Control.
  Fernando Reyes Cortés y Jaime Cid Monjaraz.
  Alfaomega Grupo Editor: "Te acerca al conocimiento".
   Continuación del sketch cap9_PIDtemperatura.ino
29 void loop(){//rutina de lazo principal de programación.
       if (bandera){//protocolo de comunicación con MATLAB.
30
          tk=0;//tiempo inicial discreto t_k = 0.
31
          lim_sup=1.04*Td; lim_inf=0.7*Td;
32
          Bp=lim_sup-lim_inf;
33
          kp=100/Bp;
34
          cota_max=kp*Td;//condición inicial \tilde{e}(0) = T_d.
35
36
          do{//transmisión/recepción de comandos de sincronía en la comunicación.
             Serial.print(33.0000,4);
37
             Serial.print(" ");
38
             Serial.print(h,4);
39
             Serial.print(" ");
40
             Serial.println(base_tiempo/h,4);
41
             if(Serial.available()>0){//disponibilidad de comando?
42
                fase=Serial.read();//lee comando desde MATLAB.
43
                if (fase==95){//limpia registro de comunicación serial.
44
                   bandera=0;//linpia bandera de protocolo de comunicación.
45
                   Serial.print(0);
46
                   Serial.print(" ");
47
                   Serial.print(0);
48
                   Serial.print(" ");
49
                   Serial.println(0);
50
                   break;
51
                }
52
             }
53
             delay(600);
54
           while (1);
55
56
```

328 Control

```
Continúa código Arduino 9.8b: sketch cap9_PIDtemperatura
   Arduino. Aplicaciones en Robótica y Mecatrónica.
   Capítulo 9 Control.
   Fernando Reyes Cortés y Jaime Cid Monjaraz.
   Alfaomega Grupo Editor: "Te acerca al conocimiento".
   Continuación del sketch cap9_PIDtemperatura.ino
57
        if(tk<br/>base_tiempo){//Algoritmo de control PID u_{PID}(t_k).
           Temperatura=125.0*5.0*analogRead(A0)/1024.0;//adquisición de datos.
58
           error=Td-Temperatura;//error de temperatura \tilde{e}(t) = T_d - T(t_k).
59
           Intk=Intk+h*error;//integral trapezoidal discreta: \int_0^t \tilde{e}(\sigma)d\sigma.
60
           //Estimación de la velocidad por el método de Euler: \frac{T(t_k)-T(t_{k-1})}{h}.
61
           euler=(Temperatura-Temperatura-a)/h;
62
           //Control PID discreto: u_{\text{PID}} = k_p \left[ \tilde{e}(t_k) + \frac{1}{t_i} \int_0^t \tilde{e}(\sigma) d\sigma - t_d \frac{T(t_k) - T(t_{k-1})}{h} \right].
63
           u_PID=kp^*(error+(1/ti)^*Intk-td^*euler);
64
65
           Temperatura_a=Temperatura; //actualiza temperatura anterior: T(t_{k-1}) = T(t_k).
66
           envia_datosMatlab(tk,Temperatura,u_PID);//graficar en MATLAB.
67
           tk=tk+h;//evolución de la variable tiempo: t_k=kh.
           /* Convierte u_PID a modulación por ancho de pulsos (PWM).*/
68
69
           Convierte_PID_pulsos_PWM(u_PID);
70
        }
71 }
```

El script **cap9_Graficar_Con_Matlab** (ver cuadro de código **MATLAB** 9.7) es el mismo programa de los ejemplos previos que se ejecuta en **MATLAB** para recibir la información de la temperatura y desplegarla en forma gráfica, en este caso recibe 90,000 puntos para formar la gráfica durante 15 minutos de controlar el horno, el usuario puede modificar el código para que grafique de manera indefinida.

La figura 9.18 muestra el comportamiento monótono creciente de la temperatura del interior del horno, el límite inferior de la banda proporcional se ha definido del 70 % de la referencia, es decir: lim_inf=0.7 250 °C = 175 °C, mientras que el límite superior del 0.04 % por encima de T_d , en otras palabras: lim_sup=260 °C. Por lo tanto, la banda proporcional B_p seleccionada es de 85 °C. Observe que se ha considerado un sobreimpulso máximo del 0.04 % de T_d , el cual corresponde a 10 °C, de acuerdo a los resultados experimentales el máximo sobretiro fue de 8.72 °C. El tiempo derivativo t_d es sintonizado en 8 mseg, para tener un efecto de anticipación adecuado en la respuesta del horno y de esta forma obtener un efecto de amortiguamiento o freno mecánico que permita alcanzar un

sobreimpulso menor al contemplado (8.72 °C < 10 °C).

El tiempo integral t_i es seleccionado en 300 segundos, para no interferir en el transitorio; sin embargo, con el tiempo necesario (5 minutos) para actuar cuando el error de temperatura $\tilde{e}(t_k)$ es negativo, y de esta forma en la bajada del sobreimpulso llevarlo lentamente en la región de estabilización alrededor de la temperatura de referencia (250 °C) convergiendo a un valor estacionario de 249.9 °C.

Observe que en la respuesta del horno eléctrico de la figura 9.18 se muestra un perfil suave en el tiempo, sin ruido, ni rizo u oscilaciones, lo anterior obedece a una adecuada interface electrónica de acoplamiento del sensor de temperatura (termopar), compensador de temperatura ambiente, etapa de potencia con cruce por cero y una correcta implementación del algoritmo de control PID.

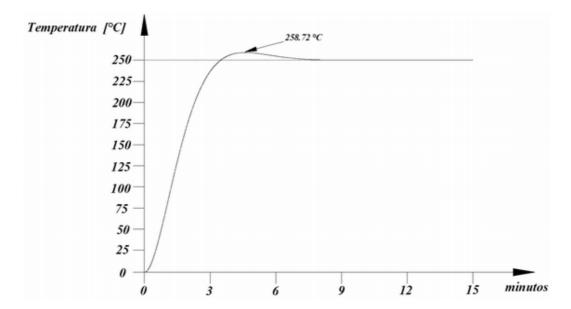


Figura 9.18 Temperatura experimental del horno eléctrico.

La ejecución del sketch **cap9_PIDtemperatura** en la tarjeta Arduino es similar al descrito en la página 298.

330 Control



9.4 Resumen

E la sistema Arduino representa una importante herramienta para automatizar una variedad de procesos que se emplean en ingeniería y ciencias exactas. Las características electrónicas de las tarjetas Arduino permiten acoplar la interface adecuada de acuerdo a los requerimientos de la planta física o proceso.

Un aspecto invaluable del sistema Arduino es la arquitectura abierta; desde el punto de vista técnico se pueden acoplar al puerto digital circuitos especializados como FPGA's, timers, convertidores digital/analógico y analógico/digital y llevar a cabo la programación en lenguaje C para adquirir adecuadamente la información.

En el presente capítulo se ha puesto de manifiesto el poder de cómputo que tienen las tarjetas Arduino, por ejemplo realizan perfectamente algoritmos complejos como la solución discreta de la ecuación (9.3). Los resultados de cómputo fueron comparados con los que entrega el ambiente de programación MATLAB.

La descripción completa del control de temperatura de un horno eléctrico para fundido y moldeo de plástico ha sido discutido utilizando la estrategia proporcional integral derivativo (PID) con la sintonía de las ganancias.

La instrumentación electrónica que acopla la señal del termopar y su compensador de temperatura, así como el envío del comando de control a la etapa de potencia resultan factores claves para obtener eficiencia y rendimiento en la respuesta del horno.



9.5 Referencias selectas

A siguiente bibliografía se recomienda al lector para abundar los temas de control PID, sistemas discretos y aspectos prácticos de implementación.



Karl J. Åström & Björn Wittenmark. "Computer Controlled Systems: Theory and Design". Prentice-Hall, Editor Thomas Kailath, Third Edition. 1997.



Fernando Reyes Cortés. "MATLAB Aplicado a Robótica y Mecatrónica". Alfaomega Grupo Editor. 2012.



9.6 Problemas propuestos

En esta sección se presenta un conjunto de problemas para mejorar el grado de comprensión de los temas y conceptos tratados en este capítulo.

A continuación se desglosan diversos planteamientos con diferentes tipos de retos técnicos y teóricos que pretenden mejorar los ejemplos desarrollados para sistemas de segundo orden y control de temperatura.

- 9.6.1 Del ejemplo 9.1 proponga un procedimiento en **MATLAB** para calcular el valor de los coeficientes α_0 y α_1 .
 - a) Tomando en cuenta la matriz $A \in \mathbb{R}^{2 \times 2}$ del sistema de segundo orden realice un script en MATLAB para obtener los valores propios.
 - b) Plantee un procedimiento en MATLAB para resolver el sistema de ecuaciones (9.15a)-(9.15b).
 - c) ¿Cuál sería la programación en MATLAB para calcular las matrices Φ y Γ ?
- 9.6.2 En el ejemplo 9.1 se obtuvieron las matrices Φ y Γ del sistema discreto, particularmente en la página 288 se calculó la integral $\Gamma = \int_0^h e^{Ah} B dh$:
 - a) ¿Qué método numérico puede utilizar para calcular la integral en forma recursiva tal que proporcione el mismo resultado de la expresión (9.25)?
 - b) ¿Cómo selecciona el intervalo de integración?
 - c) ¿Cuál sería el código de programación en lenguaje C?
- 9.6.3 En el ejemplo 9.1 llevar a cabo la simulación con los siguientes factores de amortiguamiento (mantener el valor de la frecuencia natural de resonancia $w_n = 1 \text{ rad/seg}$):
 - a) $\rho = 0$.
 - b) $\rho = 0.003$.
 - c) $\rho = 0.99$.
 - d) $\rho = 1.34$.

Analizar y discutir los resultados de cada caso.

332 Control

9.6.4 En el ejemplo 9.2 realice la simulación en **MATLAB** y en lenguaje C (con la ejecución del correspondiente script en la tarjeta Arduino) del sistema dinámico lineal de segundo orden con las siguientes estrategias de control:

- a) $u(t) = k_p \arctan(\tilde{x}(t)) k_v \arctan(x_2(t))$.
- b) $u(t) = k_p \operatorname{senh} \tilde{x}(t) k_v \operatorname{senh} x_2(t)$.

c)
$$u(t) = k_p \frac{\tilde{x}(t)}{\sqrt{1+\tilde{x}^2(t)}} - k_v \frac{x_2(t)}{\sqrt{1+x_2^2(t)}}$$
.

conserve en todos los casos los valores de ρ , w_n , k_p , k_v , x_d y h. Verifique que los resultados de simulación obtenidos en MATLAB corresponden a los simulados en la tarjeta Arduino.

- 9.6.5 ¿Qué tipos de sensores (diferente al LM35) puede utilizar para llevar a cabo la compensación de temperatura ambiente de un termopar JK-chromel-alumel?
- 9.6.6 Seleccione un amplificador de instrumentación para acoplar la señal del termopar utilizando un arreglo de amplificadores operacionales (LM747).
- 9.6.7 En referencia al ejemplo 9.4, controle la temperatura de un horno eléctrico con las siguientes leyes de control:
 - a) $u_{\text{PID}} = k_p \left[\operatorname{atan} \left(\tilde{e}(t) \right) + \frac{1}{t_i} \int_0^t \operatorname{atan} \left(\tilde{e}(\sigma) \right) d\sigma t_d \operatorname{atan} \left(\dot{T}(t) \right) \right].$
 - b) $u_{\text{PID}} = k_p \left[\tanh(\tilde{e}(t)) + \frac{1}{t_i} \int_0^t \tanh(\tilde{e}(\sigma)) d\sigma t_d \tanh(\dot{T}(t)) \right].$
 - c) $u_{\text{PID}} = k_p \left[\operatorname{senh}(\tilde{e}(t)) + \frac{1}{t_i} \int_0^t \operatorname{senh}(\tilde{e}(\sigma)) d\sigma t_d \operatorname{senh}(\dot{T}(t)) \right].$

Considere una banda proporcional $B_p = 1.05 \% T_d$ -0.7 % T_d , la temperatura de referencia $T_d = 200 \,^{\circ}$ C. Sintonice el tiempo integral $t_i = 300 \,^{\circ}$ Segundos y $t_d = 0.001 \,^{\circ}$ Segundos.

9.6.8 En el cuadro de código Arduino 9.8 modifique la programación del algoritmo de tal manera que no se requiera utilizar la función:

Convierte_PID_pulsos_PWM(u_PID)

en su lugar utilice el puerto digital pin3 para generar la señal PWM del sistema Arduino, usando la función **analogWrite(...)**. ¿Es necesario configurar el puerto digital pin 3 como salida?

Capítulo



- 10.1 Introducción
- 10.2 Bluetooth
- 10.3 Librerías para comunicación serial
- 10.4 Bluetooth Arduino+MATLAB
- 10.5 Resumen
- 10.6 Referencias selectas
- 10.7 Problemas propuestos

Competencias

Presentar las herramientas de comunicación inalámbrica de Bluetooth para desarrollar aplicaciones de mecatrónica, robótica y en general en las áreas de las ingenierías y ciencias exactas. Se describen ejemplos ilustrativos que permiten comunicar inalámbricamente dispositivos remotos con Sistema Android (teléfonos celulares y tablets) y computadoras con el Sistema Operativo Windows. También se desarrollan aplicaciones de sistemas discretos y control digital con librerías para comunicación Bluetooth en Arduino y MATLAB.

Desarrollar habilidades en:

- Programación en lenguaje C para enlazar dispositivos remotos con comunicación Bluetooth.
- Enlace de comunicación inalámbrica con teléfonos móviles, tablets y computadoras.
- Funciones Arduino para Bluetooth.
- Funciones MATLAB para comunicación inalámbrica.
- Desarrollo de aplicaciones en control automático con Bluetooth (MATLAB + Arduino).

10.1 Introducción



 ${
m B}^{
m LUETOOTH}$ es una especificación industrial para redes inalámbricas de área personal (WPAN) que facilita la comunicación de voz y datos entre diferentes dispositivos mediante enlace por radiofrecuencia en la banda de 2.4 GHz.

Los dispositivos que utilizan la transmisión de datos en forma inalámbrica pertenecen a los sectores de telecomunicación como son las compañías radiodifusoras y televisoras, telefonía móvil (celulares), computadoras, impresoras, cámaras digitales, mouse, apuntadores laser, entre otros más.



Bluetooth

Bluetooth es un protocolo de transmisión de datos en forma inalámbrica diseñado especialmente para dispositivos de bajo consumo, que requieren corto alcance de emisión a bajo costo.

La comunicación Bluetooth tiene las siguientes ventajas:



Permite la comunicación entre dispositivos móviles.



No requiere de cables o conectores.



Se pueden diseñar pequeñas redes inalámbricas entre equipos personales.



Los dispositivos que incorporan este protocolo pueden comunicarse entre ellos, si la potencia de transmisión es suficiente y cuando se encuentran dentro del rango de alcance. Esta tecnología utiliza radiofrecuencia lo que facilita el enlace, ya que no requiere que los dispositivos estén alineados.

De acuerdo a su potencia de transmisión, los dispositivos Bluetooth se clasifican en tres clases como se indica en la tabla 10.1. La cobertura efectiva de un dispositivo de clase 2 puede ser extendida cuando se conecta a un transceptor de clase 1. Las características de sensibilidad de los dispositivos clase 1 permiten recibir la señal de otros a pesar de ser más débil.

Tipo de clase Potencia máxima permitida Alcance aproximado mWdBm100 mW $\sim 30 \text{ m}$ 1 20 dBm2 $2.5~\mathrm{mW}$ $\sim 5~\mathrm{a}~10~\mathrm{m}$ 4 dBm 3 1 mW0 dBm $\sim 1 \text{ m}$

Tabla 10.1 Clasificación de dispositivos Bluetooth.



Breve historia 👔



Bluetooth es la tecnología comercial y popular del estándar de comunicación inalámbrica IEEE 802.15.1; permite la comunicación de datos sin cables, ni conectores, para crear pequeñas redes domésticas que permitan sincronizar y compartir información almacenada entre diversos equipos. El nombre Bluetooth procede del rey danés y noruego Harald Blåtand (935 D. C.), traducido como Harold Bluetooth conocido por unificar al cristianismo a las tribus noruegas, suecas y danesas. La idea de este nombre fue propuesto por Jim Kardach en 1996 quien desarrolló un sistema que permite enlazar teléfonos móviles con computadoras y unificar la comunicación en sistemas digitales. El logo de Bluetooth 🔀 representa las iniciales del nombre Hagall y apellido Berkana, proviene de símbolos rúnicos que forman un tipo de alfabeto de aquella época con runas o letras para lenguas germánicas (datan del año 150 D. C.); estos símbolos se utilizaban durante la época antigua y Edad Media; fueron muy importantes durante la cristianización de la región, representaban un medio de comunicación codificado.



Desarrollo Bluetooth

La herramienta Bluetooth fue desarrollada en 1994 por Jaap Haartsen v Mattisson Sven cuando laboraban para la empresa Ericsson en Lund, Suecia, como una tecnología para susituir cables y conectores eléctricos. Los informes fueron publicados por el Bluetooth Special Interest Group (SIG), cuyo anuncio formal fue realizó el 20 de mayo de 1998. Fue creado por Ericsson, Intel, Toshiba y Nokia; actualmente cuenta con una membresía de más de 20,000 empresas en todo el mundo. Todas las versiones de los estándares de Bluetooth están diseñadas por retro-compatibilidad, esto significa que la última versión estándar cubra todas las especificaciones de las versiones previas.

Tabla 10.2 Versiones Bluetooth.

| Versión | Ancho de banda |
|-----------------|----------------|
| Versión 1.2 | 1 Mbit/s |
| Versión 2.0+EDR | 3 Mbit/s |
| Versión 3.0+HS | 24 Mbit/s |
| Versión 4.0 | 24 Mbit/s |

Otra forma de clasificar a los dispositivos Bluetooth es por medio de su ancho de banda como se indica en la tabla 10.2. La versión 1.2 es compatible con USB 1.1; la versión v2.0+EDR fue lanzada en 2004 y es compatible con la versión anterior 1.2, el sufijo EDR significa mayor velocidad de transmisión de datos (enhanced date rate), la

tasa nominal de EDR es 3 Mbit/s y proporciona menor consumo de energía. En el año del 2009 aparece la versión 3.0+HS con altas velocidades de transferencia de datos de 24 Mbit/s, mientras que la versión 4.0 se da a conocer en el 2010 dirigida para aplicaciones de muy baja potencia alimentada por medio de una pequeña batería. El Bluetooth de alta velocidad se basa en WiFi y bajo consumo; el Bluetooth clásico maneja protocolos preexistentes.



WiFi

WiFi utiliza una señal de mayor potencia en el mismo espectro de frecuencia que Bluetooth (banda de frecuencias de 2.402 a 2.480 GHz) con tecnología de espectro ensanchado por saltos de frecuencia (FHSS Frequency Hopping Spread Spectrum); ambas tecnologías operan en las bandas de frecuencia no reguladas (banda ISM), consiste en dividir la banda de frecuencia en 79 canales (denominados saltos) de 1 MHz de ancho cada uno y transmitir la señal utilizando una secuencia de canales. Por lo tanto, al cambiar de canales con una frecuencia de 1600 veces por segundo, el estándar Bluetooth puede evitar la interferencia con otras señales de radio.

Dentro del ámbito de las comunicaciones inalámbricas WiFi y Bluetooth cubren necesidades distintas en diversas aplicaciones como en redes de propósito general. WiFi es similar al Ethernet tradicional, por lo que requiere de una configuración previa; Wi-Fi Direct es un programa de certificación que permite que varios dispositivos Wi-Fi se conecten entre sí sin necesidad de un punto de acceso intermedio.

Cuando un dispositivo ingresa al rango del anfitrión Wi-Fi Direct, éste se puede conectar usando el protocolo ad-hoc existente, recolectando la información de configuración usando transferencia de datos por el mismo canal de comunicación.



10.2 Bluetooth

En protocolos de comunicación de redes informáticas, el estándar Bluetooth es ampliamente utilizado ya que los nodos se conectan por medio de esta tecnología, la cual se basa en el modo de operación maestro/esclavo bajo el concepto **piconet**. Pueden coexistir hasta 10 piconets dentro de una sola área de cobertura. Un dispositivo maestro se puede conectar simultáneamente con 7 dispositivos esclavos activos (255 cuando se encuentran en modo en espera). Los dispositivos en un piconet poseen una dirección lógica de 3 bits, teniendo como máximo 8. Los dispositivos que se encuentran en el modo en espera se sincronizan, pero no tienen su propia dirección física dentro del piconet.

La figura 10.1 muestra el escenario donde un computadora portátil puede comunicarse vía Bluetooth con otros dispositivos tales como cámaras fotográficas y video, pantallas de TV, teléfonos celulares, tablets, tarjetas de instrumentación electrónicas como Arduino, etc. La información, procesamiento y presentación de datos se puede incrementar y hacer más eficiente mediante una pequeña red inalámbrica.



Figura 10.1 Piconet.



10.2.1 Arquitectura de los dispositivos Bluetooth

A grandes rasgos, el hardware de un sistema Bluetooth está formado de dos partes:



Dispositivo de radio para modular y transmitir la señal de radio-control.



Controlador digital compuesto por los siguientes dispositivos:



Un procesador de señales digitales DSP (*Digital Signal Processor*) llamado controlador de enlace (*Link Controller*), se encarga del procesamiento de la banda base y del manejo de protocolos de la capa física, así como de las funciones de transferencia en forma asíncrona y síncrona, codificación de audio y cifrado de datos.



Contiene un CPU para procesar las instrucciones Bluetooth con el dispositivo anfitrión, a través de un software denominado *link manager*, el cual facilita la comunicación con otros dispositivos.



Interfaces del dispositivo anfitrión.

Alfaomega

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

FERNANDO REYES CORTÉS • JAIME CID MONJARAZ

10.2 Bluetooth 339

Entre las tareas que realiza la unidad de control digital y el software *link manager* se encuentran las siguientes: envío y recepción de datos, paginación y peticiones, establecimiento de conexiones y tipos de enlace, transferencia de mensajes o paquetes de datos.



10.2.2 Especificaciones técnicas

La tecnología de Bluetooth opera en la frecuencias de radio dentro de la banda de 2.4 a 2.8 GHz de amplio espectro y con saltos de frecuencia con capacidad de transmitir Full Duplex (transmisión en ambos sentidos o bidireccional) con un máximo de 1600 saltos /segundo.

Los saltos de frecuencia se dan de un total de 79 frecuencias con intervalos de 1 MHz. El canal de comunicación tiene un máximo de frecuencia de 720 Kbit/s (1 Mbit/s de capacidad bruta) dentro de un rango óptimo de 10 m (se puede expandir a 100 m con repetidores).

La potencia de salida para transmitir a una distancia máxima de 10 metros es de 0 dBm (1 mW), mientras que la versión de largo alcance transmite entre 20 y 30 dBm (100 mW y 1 W, respectivamente).

Para alcanzar el objetivo de bajo consumo a menor costo se desarrolló en un chip de 9×9 mm utilizando circuitos CMOS que consume aproximadamente $97\,\%$ menos energía que un teléfono celular común.

El protocolo de banda base (canales simples por línea) combina conmutación de circuitos y paquetes; cada canal permite soportar información en datos, voz, en forma síncrona y asíncrona.

Cada canal de voz puede soportar una tasa de transferencia de 64 kbit/s en cada sentido; un canal asíncrono puede transmitir hasta 721 kbit/s en una dirección y 56 kbit/s en la dirección opuesta. Sin embargo, una conexión síncrona puede soportar 432,6 kbit/s en ambas direcciones si el enlace es simétrico.



10.2.3 Aplicaciones Bluetooth

Bluetooth se utiliza principalmente en un gran número de equipos digitales tales como computadoras, teléfonos, impresoras, módems, auriculares, automóviles, etc. Es particularmente útil cuando hay dos o más dispositivos en un área reducida sin necesidad de alta tasa de velocidad en transmisión de datos (reducido ancho de banda).



Aplicaciones Bluetooth

En el sitio Web de este libro se incluye aplicaciones de la tecnología Bluetooth para control, automatización e ingeniería robótica y mecatrónica.

La tecnología Bluetooth permite una gran cantidad de aplicaciones como las que a continuación se enlistan:

- Reemplazo de comunicación por cable entre dispositivos electrónicos.
- Controles remotos (usando señales de infrarrojo).
- Manos libres en telefonía móvil.
- Enlace inalámbrico de telefonía para automóviles.
- Comunicación con sistemas de audio/video.
- Transferencia de datos con computadoras y sistemas digitales.

En ingeniería mecatrónica y robótica, la tecnología Bluetooth se utiliza para comunicar dos o más robots manipuladores, coordinar tareas y actividades de los robots con sus sistemas periféricos tales como actuadores eléctricos y mecánicos, bandas transportadoras, compresoras, prensas, garras mecánicas; enviar y recibir comandos de control e información de sensores de posición, velocidad de movimiento, orientación de la herramienta de trabajo; autonomía y adaptación del entorno o medio ambiente donde se desenvuelve el robot a través de sensores de visión, fuerza e impedancia.

Dentro de los aspectos de automatización, se utiliza para intercambiar el valor numérico de parámetros o ganancias de algoritmos de control, así como desplegar en un monitor la información relevante al usuario que está siendo procesada en un sistema digital empotrado.

10.3 Librerías para comunicación serial



ROGRAMAS y aplicaciones comerciales de Bluetooth para dispositivos telefónicos móviles, tablets y computadoras no cumplen, ni se pueden adaptar a los requerimientos y necesidades que demanda la automatización de un proceso en particular. Por tal motivo, es necesario el diseño y desarrollo de programación acorde a las especificaciones de la aplicación a implementar; tomando esto en consideración, las áreas de ingeniería mecatrónica y robótica expanden sus potenciales impactos en la automatización de procesos utilizando plataformas electrónicas como Arduino y mediante el apoyo de los lenguajes C y MATLAB se pueden realizar aplicaciones específicas en forma ex profesa o per-se para comunicaciones inalámbricas dedicas a cubrir los requerimientos de un sistema que se va a automatizar.

A continuación se describen las funciones de la librería **SoftwareSerial** del Sistema Arduino que permiten desarrollar interfaces para comunicarse con dispositivos remotos.



10.3.1 Librería SoftwareSerial del Sistema Arduino

La librería SoftwareSerial es un conjunto de funciones para comunicación serial que permite el intercambio de información utilizando puertos digitales para realizar el intercambio de datos con velocidades hasta 115200 Baudios. Para las tarjetas Arduino es ampliamente recomendable utilizar puertos digitales diferentes a los pins 0 y 1, ya que están destinados al bloque interno UART para transmitir/recibir información a la computadora a través de la conexión USB, de esta forma se enlaza la comunicación para intercambio de datos con el monitor serial dentro de la plataforma Arduino, así como con el ambiente de programación MATLAB.

La librería SoftwareSerial tiene las siguientes limitaciones:

- Cuando se utilizan múltiples puertos seriales, sólo uno a la vez puede recibir datos, ya que no admite comunicación simultánea.
- Para los modelos Mega y Mega 2560 no todos los pins soportan cambios de interrupciones. Los siguientes pins pueden ser usados para Rx: 10, 11, 12, 13, 14, 15, 50, 51, 52, 53, A8(62), A9(63), A10(64), A11(65), A12(66), A13(67), A14(68), A15(69).
- No todos los pins del modelo Leonardo soportan cambios de interrupciones, los siguientes pins pueden emplearse para ser usados para Rx: 8, 9, 10, 11, 14(MISO), 15(SCK), 16(MOSI).

Observación: la verisón 1.0 de SofwareSerial para el sistema Arduino se basa en la librería

NewSoftSerial desarrollada por Mikal Hart (ver el sitio http://www.arduiniana.org). Dada una aplicación, si se requiere de un flujo de datos en forma simultanea, puede utilizar la librería AltSoftSerial de Stoffregen (visitar http://pjrc.com).



SoftwareSerial(Rxpin, Txpin)

Crea un nuevo objeto de clase **SoftwareSerial**, cuyo nombre es proporcionado por el usuario, es decir el resultado que retorna es un objeto de programación con las características y membresía de la librería serial. Por ejemplo:

SoftwareSerial mipuertoserial=SoftwareSerial(10, 11)

donde los argumentos de entrada de la función son **Rxpin** y **Txpin**, e indican los pins de los puertos digitales para recibir y transmitir datos, respectivamente.

Nota: de aquí en adelante y como parte de la descripción en las siguientes funciones de la librería SoftwareSerial se hará referencia sólo para propósitos ilustrativos o didácticos al objeto de clase SoftwareSerial mipuertoserial. El lector puede declarar cualquier otro nombre o identificador para dicho objeto.



SoftwareSerial: available()

Proporciona el número de bytes (carácteres) disponibles para lectura del puerto serial de clase **SoftwareSerial**. Cuando el dato se encuentra disponible, significa que se encuentra almacenado en el buffer serial de recepción. La forma de utilizarse es de la siguiente forma: **mipuertoserial.available()**; no tiene parámetros o argumentos de entrada, retorna el número de bytes disponibles para leer, esta característica se puede utilizar como un valor booleano en funciones lógicas.



SoftwareSerial: begin(Baudios)

Configura la velocidad de transmisión serial del **SofwareSerial** en Baudios (300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 31250, 38400, 57600 y 115200). El argumento de entrada **Baudios** es un dato de tipo entero largo (long), esta función no retorna información. Un ejemplo de sintaxis es la siguiente:

mipuertoserial.begin(9600);

generalmente esta función se utiliza como parte de la configuración de las tarjetas Arduino dentro de la subrutina **setup()**.



SoftwareSerial: isListening()

Detecta si una solicitud del puerto de software serial está activa. La sintaxis de esta función es: mipuertoserial.isListening(); no tiene parámetros de entrada y retorna un valor booleano verdadero si está activo; en caso contrario, falso.

SoftwareSerial: overflow()



Revisa si ha ocurrido desbordamiento o sobre-flujo (overflow) de datos en el buffer o área de memoria del puerto de software serial. Este buffer tiene una capacidad de 64 bytes y por lo tanto es muy fácil llevarlo a la saturación; una llamada a esta función limpia la bandera de overflow, por lo que en subsecuentes llamadas retornará un valor booleano falso, a menos que algún byte de datos ha sido recibido y descargado al área del buffer. La sintaxis de esta función es de la siguiente manera: mipuertoserial.overflow().

SoftwareSerial: peek()



Retorna el carácter que fue recibido en el pin Rx del puerto de software serial; también puede retornar el valor de -1 si el dato no está disponible. Subsecuentes llamadas a esta función retornará el mismo carácter, ya que trabaja diferente a la función **SoftwareSerial: read()**. Esta función no tiene argumentos de entrada. La sintaxis de esta función es la siguiente: c=mipuertoserial.peek(); siendo **c** una variable de tipo char.

SoftwareSerial: read()



Retorna el carácter que fue recibido en el pix **Rx** del puerto de software serial, también puede retornar el valor de -1 si el carácter no está disponible; esta función no tiene argumentos de entrada. Debe recordarse al lector que sólo un puerto de clase **SoftwareSerial** puede recibir datos a la vez; para seleccionar cuál puerto está recibiendo datos se emplea la función **SoftwareSerial: listen()**. Como ejemplo de sintaxis se tiene lo siguiente:

c=mipuertoserial.read();//donde c es una variable de tipo char.

SoftwareSerial: print(dato)



Transmite o imprime **dato** a traves del pin Tx del puerto de software serial; esta función trabaja de la misma forma que la función **Serial.printf()**. El argumento de entrada **dato** varía dependiendo del tipo de variable, también puede ser un comando de impresión como espacios tabulares verticales, horizontales o caracteres en blanco. Mayor información sobre la función **Serial.printf()** se puede encontrar en la página 183. Ejemplo:

mipuertoserial.printf(2345,DEC); //envía constante entera a MATLAB o terminal de datos seriales.

∞

SoftwareSerial: println(dato)

Imprime dato a través de la transmisión del pin Tx del puerto de software serial, seguido de un retorno de carro y nueva línea de texto, como parte del dato transmitido. Esta función trabaja de la misma forma que Serial.println().

El argumento de entrada **dato** puede ser de cualquier tipo, inclusive comandos de espacio tabular; para mayores detalles ver la función **Serial.println()** en la página 185. Ejemplo:

${\bf mipuertoserial.printf(2.156345,\!4);}$

Para este caso, se envía constante flotante con cuatro fracciones a **MATLAB** o alguna terminal de datos seriales.

∞

SoftwareSerial: listen()

Habilita el puerto de software serial para "escuchar" datos; sólo un puerto de software serial puede transmitir/recibir datos a la vez, y datos que lleguen a otros puertos de software serán descartados. Ejemplo: mipuertoserial.listen(); esta función no retorna datos.

∞

SoftwareSerial: write(dato)

Imprime el argumento de entrada **dato** en formato de bytes sobre el pin de transmisión Tx del puerto de software serial. Esta función trabaja de la misma forma que la función **Serial.write()** (ver página 185).



10.3.2 Módulo de Bluetooth JY-MCU

E la módulo inalámbrico JY-MCU es un dispositivo para comunicación Bluetooth con la interface electrónica para conectarse directamente a las tarjetas Arduino, computadoras personales y en general a diversos microcontroladores y sistemas empotrados.

El módulo JY-MCU viene con un conector tipo header con 6 pins (en la parte posterior del impreso se encuentra la nomenclatura de señales, ver figura 10.2); la tabla 10.3 muestra las características técnicas del módulo, el voltaje de alimentación de DC se encuentra en el rango de 3.6 V a 5 V. Este

dispositivo es compatible con la versión v2.0+EDR (enhanced data rate), la cual ofrece velocidades de 3 Mbps y Baud rate de 38400 bps; el código de acceso es "1234" y dentro de sus características importantes se destaca la relacionada a las especificaciones de conectividad por medio de Bluetooth SPP (Serial Port Protocol), por lo tanto cualquier dispositivo que soporte SPP puede conectarse (teléfonos móviles o dispositivos tablet con Sistema Android y computadoras personales con el Sistema Operativo Windows), esto excluye a dispositivos Ipad e Iphone.

Otro aspecto importante del dispositivo remoto JY-MCU, es que no requiere configuración para enlazar comunicación, solo debe estar dentro del rango de alcance, en este caso dentro de un radio de 10 metros y no requieren estar alineados, ya que se transmite por radio frecuencia. La figura 10.3 muestra la forma de realizar las conexiones eléctricas del módulo JY-MCU con la tarjeta Arduino UNO. El puerto digital pin 2 es configurado como entrada para trabajar como receptor Rx, el cual va conectado al pin Tx del dispositivo inalámbrico; mientras que el pin 3 se



Figura 10.2 Módulo JY-MCU.

configura como salida para llevar a cabo la función de transmisor Tx conectado directamente a la señal Rx del módulo remoto. En este punto, se recuerda la conveniencia de utilizar otros puertos digitales diferentes a los pins 0 y 1 de la tarjeta Arduino, ya que esos pins se relacionan con la comunicación e intercambio de información en ambos sentidos por medio del puerto USB, evitando interferencia con la programación preestablecida que configura las funciones de dichos pins.

Una vez que el dispositivo inalámbrico se encuentra ya conectado al modelo Arduino UNO (alimentado por cable USB o a través de una fuente externa) el diodo LED que está sobre el circuito impreso del módulo JY-MCU emite luz en infrarrojo indicando que se encuentra listo para llevar a cabo el enlace inalámbrico; cuando se encuentra intermitente (flashing) significa que no está conectado con algún dispositivo; por otro lado, cuando ya se encuentra enlazado, el estado visual del LED es emitiendo luz en color infrarrojo en forma permanente.

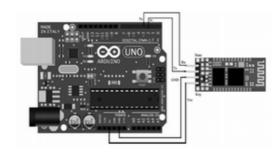


Figura 10.3 Conexiones eléctricas del módulo JY-MCU con tarjeta Arduino UNO.

Tabla 10.3 Características del módulo JY-MCU.

| Número de pin | Descripción |
|---------------|---------------------------------|
| 1 | Key |
| 2 | Vcc de 3.6 V a 6 V DC |
| 3 | GND referencia a tierra |
| 4 | Tx transmisor |
| 5 | Rx receptor |

& Ejemplo 10.1

Realizar transmisión/recepción de cadenas de caracteres (mensajes de texto) por comunicación Bluetooth entre el módulo JY-MCU y una computadora portátil con el Sistema Operativo Windows 8.1.

Solución

Para realizar la aplicación de transmisión/recepción de datos entre el módulo JY-MCU y una computadora personal con el Sistema Operativo Windows 8.1, es necesario dar de alta el módulo inalámbrico JY-MCU en la computadora personal; la forma más simple es identificando el icono de Bluetooth 💦 sobre la barra de tareas de la computadora. Aquí puede agregar dispositivos Bluetooth y su configuración, el cuadro informativo 10.1 muestra el diagnóstico del estado de conexión del dispositivo remoto Bluetooth.

Administrador de dispositivos

PC y Dispositivos

Pantalla de bloqueo Pantalla Bluetooth Dispositivos Mouse y panel táctil Escritura Esquinas y bordes Inicio/apagado y suspensión Reproducción automática Información de PC

Bluetooth Activado

PC detectando dispositivos Bluetooth...

HC-05 sin conexión



Cuadro 10.1 Estado de conexión del dispositivo remoto JY-MCU en Windows 8.1.

Otra manera de llevarlo a cabo es posicionando el mouse sobre la esquina superior derecha del

monitor, seleccione la opción Configuración y Cambiar configuración de la PC; verifique que en Red inalámbrica el servicio de Bluetooth está activado (de no ser así, habilítelo). Ahora, en Dispositivos revise que ya se encuentra el módulo inalámbrico JY-MCU identificado con el siguiente nombre "HC-05". En el cuadro 10.1 se muestra la forma en que lo presenta la computadora personal.

Establezca la conexión con el dispositivo remoto "HC-05" :

La palabra clave de acceso para el dispositivo Bluetooth JY-MCU es:

1234

El cuadro de código Arduino 10.1 describe la programación en lenguaje C del sketch cap10_Bluetooth_A para llevar a cabo la comunicación serial por software con el dispositivo remoto JY-MCU. En el header o cabecera de este programa incluye la librería SoftwareSerial.h para realizar la transferencia de datos por puerto digital de la tarjeta Arduino con el módulo Bluetooth (ver línea 1). En este caso se seleccionan los pins 2 y 3. Es recomendable no seleccionar los puertos digitales pins 0 y 1, debido a que se encuentran destinados a la comunicación serie por USB. En la línea 2 se crea el objeto tipo Bluetooth especificando al pin 2 como receptor (Rx) y al pin 3 como transmisor de datos (Tx), las conexiones eléctricas entre el módulo remoto y la tarjeta Arduino UNO se indican en la figura 10.3.

La variable **mensaje** es de tipo char para registrar las cadenas de caracteres para transmitir o recibir (línea 3). La subrutina de configuración **setup()** (línea 4) se encarga de programar la velocidad de comunicación serie por cable USB y por software serial; ambas se establecen en 9600 Baudios. El lazo principal de programación **loop()** inicia a partir de la línea 8; note que en la línea 9 se determina disponibilidad de información en el dispositivo remoto Bluetooth, si esta condición es verdadera, entonces se lee el puerto serial Bluetooth (línea 10) y se transmite dicha información por cable USB (línea 11) al monitor serial del ambiente Arduino para su presentación en formato de texto.

En la computadora personal se requiere instalar un programa que permita transmitir/recibir mensajes de texto por puerto serial inalámbrico. Existen varios programas que funcionan como puerto o terminal virtual Bluetooth para computadoras personales con Sistema Operativo Windows; dentro de ellos, la interfaz **Hercules** es muy fácil de utilizar. La figura 10.4 muestra la ventana principal con los menús de configuración que tiene este programa, el cual es un producto diseñado por HWgroup y es de uso libre; se puede descargar de la siguiente dirección electrónica:



Figura 10.4 Hercules.

http://www.hw-group.com/

Cuando el usuario transmite una cadena de caracteres desde el monitor serial Arduino, entonces la condición sobre la línea 13 es verdadera, esa cadena se registra en la variable **mensaje** por medio de **Serial.read()** por cable USB (línea 14) y por software serial **Bluetooth.print(mensaje)** se envía al dispositivo remoto como se muestra en la línea 15.

```
○ Código Arduino 10.1: sketch cap10_Bluetooth_A
  Arduino. Aplicaciones en Robótica y Mecatrónica.
  Capítulo 10 Bluetooth.
  Fernando Reyes Cortés y Jaime Cid Monjaraz.
  Alfaomega Grupo Editor: "Te acerca al conocimiento".
   Sketch cap10_Bluetooth_A.ino
 1 #include <SoftwareSerial.h>//librería para comunicación serie por software.
 2 SoftwareSerial Bluetooth(2,3); //Arduino software serial: pin2=Rx y pin3=Tx.
 3 char mensaje;//variable que registra cadenas de caracteres o mensajes de texto.
 4 void setup(){
       Serial.begin(9600); //velocidad comunicación Serial USB: pins 0 y 1.
       Bluetooth.begin(9600); //velocidad del puerto serial por software: pins 2 y 3.
 6
7 }
8 void loop(){
       if(Bluetooth.available()>0){ //¿Dato disponible en puerto serial Bluetooth?
          mensaje = Bluetooth.read(); //lectura del puerto serial Bluetooth.
10
         Serial.print(mensaje); //transmisión por puerto serial al monitor serial
11
   Arduino.
12
       if(Serial.available()>0){ //¿Dato disponible en el monitor serial?
13
          mensaje = Serial.read(); //lectura de información.
14
          Bluetooth.print(mensaje); //transmisión por puerto serial Bluetooth.
15
16
       }
17 }
```

Para configurar la interface **Hercules** hay que referirse al menú principal, seleccione la opción **serial** y elija el recuadro **name** para asignar tipos de puertos disponibles, esto depende de los recursos y características de la computadora personal, dicha información se encuentra disponible en el **panel**

de control, sección de administrados de dispositivos, puertos LPT y COM; como un ejemplo ilustrativo, considere el caso que se muestra en la tabla 10.4, donde el puerto COM6 se usa para comunicación por cable USB con la tarjeta Arduino; adicionalmente se cuentan con dos puertos Bluetooth (COM3 y COM5).

Observe que inicialmente se asignó el puerto COM3 marcando error al intentar abrir el canal de comunicación tal y como se indica en el área de mensajes de la interface **Hercules** (ver figura 10.5), el puerto aceptado fue el COM5.

La figura 10.5 muestra los resultados de comunicación Bluetooth entre el módulo remoto JY-MCU y la interface **Hercules**; este enlace es en ambas direcciones, es decir, el usuario puede enviar mensajes de texto desde el programa de interface y visualizarlos en el monitor serial, así como transmitir datos desde el ambiente de programación Arduino y exhibirlos sobre el programa **Hercules**.

Tabla 10.4 Información del administrador de dispositivo puertos: COM y LPT.

| 7 | PUERTOS (COM y LPT) | |
|---|---------------------|---|
| | 7 | Arduino UNO R3 (COM6) |
| | 7 | Serie estándar sobre vínculo Bluetooth (COM3) |
| | 7 | Serie estándar sobre vínculo Bluetooth (COM5) |

El procedimiento para ejecutar correctamente este ejemplo se describe a continuación:

- Editar el sketch cap10_Bluetooth_A del cuadro de código Arduino 10.1.
- Compilar el sketch cap10_Bluetooth_A mediante el icono 🕢.
- Descargar el código de máquina a la tarjeta Arduino usando 👈.
- Abrir la ventana del monitor serial en el ambiente Arduino usando el icono 🔑.
- Ejecutar la interface **Hercules**.

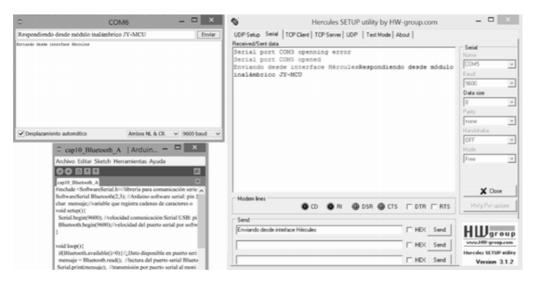
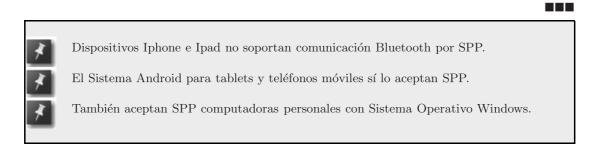


Figura 10.5 Comunicación Bluetooth entre módulo JY-MCU e interface Hercules.

- En la opción **Serial**, configurar el puerto en el recuadro **name**. La información técnica de su computadora personal se encuentra disponible en el **panel de control**, administrador de dispositivos, puertos (COM y LPT).
- Escribir un texto de prueba en la interface **Hercules** y oprimir **Send**.
- Comprobar que si se recibe el mensaje de prueba directamente en el monitor serial del ambiente Arduino.
- Desde el monitor serial, escribir algún mensaje de prueba \leftarrow y verificar que en el área de mensajes del programa **Hercules** se despliega correctamente la cadena de caracteres transmitida.







(a) Telefonía móvil.

(b) Tablet.

Figura 10.6 Dispositivos con Sistema Android.

🌲 👫 Ejemplo 10.3

Transmitir y recibir cadenas de caracteres por comunicación serial Bluetooth entre un teléfono móvil con Sistema Android y el módulo inalámbrico JY-MCU.

Solución

El sktech cap10_Bluetooth_A se encuentra en el cuadro de código Arduino 10.1 y contiene la programación requerida para acceder al módulo remoto JY-MCU (ver figura 10.3 para conexiones eléctricas).

Como un caso de estudio, considere el teléfono móvil de la compañía Samsung, modelo Galaxy 3 mini con Sistema Android; para este celular se requiere descargar una aplicación que funcione como terminal virtual de comunicación serial Bluetooth con soporte SSP.

Existe una gran cantidad de aplicaciones gratuitas para Android, entre ellos la aplicación BT Term Beta se puede descargar directamente de Play Store. La figura 10.7a muestra la aplicación BT Term la cual permite enlazar con el módulo remoto, y la figura 10.7b describe el monitor serial del ambiente Arduino. La transmisión y recepción de mensajes de texto es bidireccional entre ambos sentidos de los dispositivos conectados.





- (a) Teléfono celular.
- (b) Monitor serial Arduino.

Figura 10.7 Transmisión y recepción Bluetooth entre celular y módulo JY-MCU.

♣ ♣ Ejemplo 10.3

Transmitir y recibir mensajes de texto por Bluetooth entre el módulo remoto JY-MCU y una tablet con Sistema Android.

Solución

Para dar solución al problema planteado, se hace referencia a la programación del módulo remoto JY-MCU contenida en el sketch **cap10_Bluetooth_A** descrito en el cuadro de código Arduino 10.1, y a la figura 10.3 para conexiones eléctricas. Para finalidades prácticas, se ha seleccionado el siguiente modelo de la compañía Samsung: tablet Galaxy Note 8.0 GT 5110 con el Sistema Android 4.2.2. Para este modelo se descarga la aplicación gratuita **BlueSerial Beta** de Play Store.

El procedimiento para ejecutar el proceso de comunicación bilateral inalámbrica es el siguiente: ejecutar el sketch **cap10_Bluetooth_A** en la tarjeta Arduino y abrir el monitor serial para transmitir y recibir mensajes de datos. Por otro lado, en la tablet ejecutar la aplicación **BlueSerial**, lo primero que hace este programa es buscar dispositivos Bluetooth dentro del radio de alcance para realizar enlace.

La figura 10.8 muestra el escenario de comunicación Bluetooth en ambas direcciones entre el dispositivo tablet y el módulo remoto JY-MCU. Las aplicaciones en automatización, mecatrónica y

robótica a través de esta tecnología no sólo se amplía, también se diversifica el rango de cubertura a todo tipo de procesos.





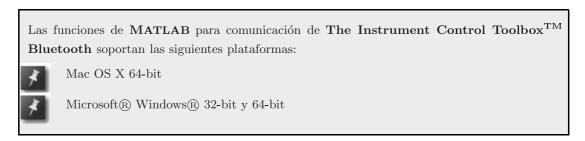
- (a) Aplicación BlueSerial en Tablet.
- (b) Monitor serial Arduino.

Figura 10.8 Transmisión y recepción Bluetooth entre tablet y módulo JY-MCU.



10.3.3 Funciones de puerto serial Bluetooth de Matlab

MATLAB cuenta con funciones específicas para comunicación inalámbrica del tipo Bluetooth denominado The Instrument Control ToolboxTM Bluetooth permitiendo conectar dispositivos sobre interfaces Bluetooth para transmitir/recibir datos ASCII y binarios. Estas funciones soportan el protocolo sobre puerto serial (SPP) para Bluetooth, por lo que interfaces como el módulo remoto JY-MCU pueden ser conectados para desarrollo de aplicaciones.



A continuación se describe la sintaxis de las funciones de MATLAB para comunicación Bluetooth.



Construye un objeto Bluetooth asociado a 'RemoteName' y con las características definidas en canal. En otras palabras, retorna un objeto de programación de clase Bluetooth; 'RemoteName' es el nombre amigable que tiene el dispositivo remoto, corresponde a una cadena de caracteres.

Por ejemplo, para identificar a un teléfono celular puede ser con el nombre 'micel', 'myphone' o cualquier identificador que tenga prestablecido el usuario, mientras que el argumento **canal** indica el número de canales asociados a dicho dispositivo. Si el argumento **canal** no se especifica, por omisión será considerado como 0.

Si la cadena de caracteres de 'RemoteName' se encuentra vacía, entonces se deberá utilizar 'RemoteID', por tal motivo la función **Bluetooth(...)** también admite la siguiente forma de sintaxis:



Bluetooth('RemoteID', canal)

Construye y por lo tanto retorna un objeto de clase Bluetooth de acuerdo a los argumentos 'RemoteID' y **canal**. La propiedad del objeto Bluetooth 'RemoteID' corresponde a un identificador interno, cada dispositivo tiene uno y usualmente es una cadena de texto de 12 dígitos que empieza con btspp:// como en el siguiente ejemplo: 'btspp://001208019224' .

La propiedad del objeto Bluetooth 'RemoteName' puede estar vacía y en tal caso deberá usarse 'RemoteID' .

Para encontrar dispositivos Bluetooth se utiliza la función **instrhwinfo(...)** con las siguientes formas de sintaxis:



instrhwinfo('Bluetooth')

Retorna una estructura de datos con la información de los dispositivos inalámbricos que se encuentran en la periferia 'Bluetooth', indicando la versión de toolbox, software de **MATLAB**, el tipo de soporte para interface, así como la naturaleza de los adaptadores. Para puertos seriales la información incluye disponibilidad y nombre del objeto constructor.

La búsqueda de dispositivos Bluetooth puede ser más rápida si se proporciona el **RemoteName**, en tal caso se emplea la siguiente sintaxis:



instrhwinfo('Bluetooth', RemoteName)

Para las sintaxis **Bluetooth** ('RemoteName', canal) y en su variante de la forma **Bluetooth** ('RemoteID', canal), la comunicación del objeto Bluetooth construido se encuentra cerrada, es decir,

se requiere abrir y conectar el enlace de comunicación a través de la función **fopen**(...) como a continuación se describe para el caso del módulo JY-MCU, el cual se asume que está conectado a la tarjeta Arduino UNO como se indica en la figura 10.3.

En la ventana de comandos de MATLAB teclee lo siguiente:

```
\begin{array}{ll} \textit{fx} >> & \text{instrhwinfo('Bluetooth' , 'HC-05' )} & \hookleftarrow \\ \textit{fx} >> & \text{ans=} \\ & \text{RemoteName: 'HC-05'} \\ & \text{RemoteID: 'btspp://001208019224'} \\ & \text{ObjectConstructorName: \{'Bluetooth('HC-05' , 1);' \}} \\ & \text{Channels: \{'1'\}} \end{array}
```

La siguiente función abre y realiza el enlace entre la computadora y un dispositivo remoto:

fopen(dispositivo_Bluetooth)



Conecta el **dispositivo_Bluetooth** para iniciar enlace de comunicación inalámbrica entre el dispositivo remoto y **MATLAB**.

El argumento de entrada se refiere al objeto construido usando la función **Bluetooth(...)** de la siguiente forma:

dispositivo_Bluetooth = Bluetooth('HC-05',1);

fwrite(dispositivo_Bluetooth, dato)



Cuando el enlace de comunicación inalámbrica ya se encuentra abierto para el **dispositi-**vo_Bluetooth a través de la función fopen(...), entonces es posible transmitir/recibir datos en
forma bidireccional. Para transmitir un mensaje desde MATLAB se emplea la función fwrite(...),
donde dato es cualquier tipo de variable entera, char, flotante o cadena de mensaje de texto.

La información que envía el dispositivo remoto se recibe en MATLAB por la siguiente función:

dato_recibido=fread(dispositivo_Bluetooth, num_bytes)



Esta función retorna la información transmitida por el dispositivo remoto en dato_recibido, el

argumento de entrada **num_bytes** indica el número de bytes que se leerán del buffer de recepción serial Bluetooth.

Otra forma para leer la información que transmite el dispositivo remoto es por medio de la siguiente función:



dato_recibido=fscanf(dispositivo_Bluetooth, comando_formato)

Retorna en **dato_recibido** la información que transmite el dispositivo remoto, lee y convierte datos de acuerdo al tipo de formato definido por **comando_formato**.

La información que retorna $\mathbf{fscanf}(...)$ también puede ser en forma de vector \mathbf{x} con r renglones y c columnas, es decir: $\mathbf{x} \in \mathbb{R}^{r \times c}$; los elementos de \mathbf{x} son del tipo $\mathbf{comando_formato}$.

Para este caso, esta función admite otro tipo de sintaxis que a continuación se describe:



x=fscanf(dispositivo_Bluetooth, comando_formato, [r, c])

La función **fscanf(...)** con este último formato de sintaxis resulta apropiada para realizar adquisición de datos en aplicaciones de ingeniería, mecatrónica, robótica y en general en todas las áreas de la automatización.

Antes de finalizar la aplicación de comunicación Bluetooth es recomendable desconectar al dispositivo_Bluetooth usando la siguiente función fclose(...) de la siguiente manera:



fclose(dispositivo_Bluetooth)

También es conveniente limpiar y borrar el área de memoria asignada al **dispositivo_Bluetooth** por medio de la función:



clear dispositivo_Bluetooth

10.4 Bluetooth Arduino+Matlab



Para incrementar, mejorar y obtener mejor eficiencia en aplicaciones de los ejemplos Bluetooth de ingeniería.

Por medio de las funciones de la librería **SoftwareSerial** se pueden diseñar aplicaciones de adquisición de datos, control de sistemas mecatrónicos y automatización de procesos enviando la información sin la necesidad de cables; la presentación de datos en forma gráfica se puede realizar en **MATLAB** recibiendo esa información usando las funciones del **The Instrument Control Toolbox**TM **Bluetooth**.

En esta sección se desarrollan ejemplos prácticos que muestran la utilidad del enlace bidireccional (transmisión/recepción en ambas direcciones) entre la tarjeta Arduino con el módulo inalámbrico JY-MCU y MATLAB. Se presentan dos ejemplos específicos, el primero es la transmisión y recepción de mensajes de cadena de texto y el segundo es la simulación de un proceso numérico de control digital en la tarjeta Arduino y la presentación de resultados en MATLAB.

La figura 10.9 muestra la idea central para comunicar tarjetas electrónicas Arduino con el ambiente de programación MATLAB.



Figura 10.9 Comunicación Bluetooth por medio de Matlab y Arduino.

♣ ♣ Ejemplo 10.4

Desarrollar una aplicación de transferencia/recepción de cadenas de texto entre el ambiente de programación MATLAB y la tarjeta Arduino UNO con el módulo remoto JY-MCU.

Solución

El cuadro de código Arduino 10.2 contiene la descripción y documentación técnica del sketch cap10_Bluetooth_B para realizar la comunicación Bluetooth bidireccional de mensajes de texto o cadenas de caracteres entre la tarjeta Arduino UNO con el ambiente del programación MATLAB. Las conexiones eléctricas del módulo JY-MCU y la tarjeta Arduino se muestran en la figura 10.3.

En el header del sketch cap10_Bluetooth_B se incluye la librería SoftwareSerial, como se indica en la línea 1; el objeto constructor del objeto_remoto de clase SofwareSerial se declara en la línea 2, definiendo la parte de las conexiones eléctricas entre la tarjeta Arduino UNO con el módulo JY-MCU: el puerto 2 corresponde al receptor Rx (aquí se conecta el pin Tx del módulo remoto) y el transmisor Tx definido en el puerto 3, el cual se conecta al pin Rx del módulo inalámbrico. Dentro de la subrutina de configuración Setup() se define la velocidad de transmisión en 9600 Baudios para comunicación serial usando el cable USB entre la tarjeta Arduino y la computadora (línea 5, mientras que la velocidad serial entre la tarjeta Arduino y el módulo Bluetooth (por software serial) se establece en el mismo valor de 9600 Baudios (línea 6).

Cuando MATLAB transmite inalámbricamente una cadena de texto (por el puerto Bluetooth de la computadora), esta información la recibe el módulo JY-MCU, el cual envía ese mensaje por el pin Tx conectado al pin 2 de la tarjeta Arduino que funciona como receptor Rx; la disponibilidad de datos seriales en el puerto inalámbrico se lleva a cabo por software en la línea 9 empleando la función objeto_remoto.available(), si el valor retornado es verdadero, entonces la cadena de mensajes se recibe en la variable c de tipo char a través del puerto serial por software utilizando la función objeto_remoto.read(); tal y como se indica en la línea 10, el mensaje puede ser desplegado en el monitor serial del ambiente de programación Arduino, ya que se utiliza la función Serial.print(c) declarada sobre la línea 11.

De la misma forma, cuando el usuario envía una cadena de texto a través del monitor serial del ambiente de programación Arduino, la programación de la línea 13 habilita la lectura por comunicación USB para registrar la cadena de caracteres usando la función **Serial.read()** como se ilustra en la línea 14; la información de esa cadena se envía al módulo JY-MCU a través del pin 3 Tx empleando **objeto_remoto.print(c)** (línea 15), dicho módulo Bluetooth la transmite inalámbricamente por señal de radio hacia **MATLAB** para su presentación en la ventana de comandos.

```
⊙ Código Arduino 10.2: sketch cap10_Bluetooth_B
  Arduino. Aplicaciones en Robótica y Mecatrónica.
  Capítulo 10 Bluetooth.
  Fernando Reyes Cortés y Jaime Cid Monjaraz.
  Alfaomega Grupo Editor: "Te acerca al conocimiento".
   Sketch cap10_Bluetooth_B.ino
 1 #include <SoftwareSerial.h>//librería para comunicación serie por software.
 2 SoftwareSerial objeto_remoto(2,3); //pin 2 es Rx, pin 3 es Tx en el Software Serial.
 3 char cadena;//variable que registra cadenas de caracteres o mensajes de texto.
 4 void setup(){
       Serial.begin(9600); //velocidad comunicación Serial USB: pin 0 y 1.
       objeto_remoto.begin(9600); ///velocidad del puerto serial por software: pin 2 y 3.
 7 }
 8 void loop(){
       if(objeto_remoto.available()>0){ //i Dato disponible en puerto serial Bluetooth?
10
          cadena = objeto_remoto.read(); //lectura del puerto serial Bluetooth.
11
          Serial.print(cadena); //transmisión por puerto serial al monitor serial Arduino.
12
       if(Serial.available()>0){ //¿Dato disponible en el monitor serial?
13
          cadena = Serial.read(); //lectura de información.
14
          objeto_remoto.print(cadena); //transmisión por puerto serial Bluetooth.
15
16
17 }
```

Por otro lado, el script cap10_Bluetooth_Matlab_B se ejecuta directamente desde MATLAB, su descripción se encuentra en el cuadro de código 10.3. El enlace de comunicación se estable por medio de fopen(dispositivo_Bluetooth) (ver línea 7). Para obtener la información específica del módulo de comunicación Bluetooth JY-MCU se realiza en la línea 5 a través de la función instrhwinfo(...), la construcción del objeto clase Bluetooth se lleva a cabo en la línea 6 usando la función dispositivo_Bluetooth = Bluetooth('HC-05',1). De aquí en adelante la comunicación Bluetooth entre el módulo JY-MCU y MATLAB ha quedado abierta. Como mensaje de prueba desde MATLAB se envía una cadena de caracteres (linea 8) a través de la función fwrite(...) como se define sobre la línea 9; este mensaje aparece en el monitor serial del ambiente Arduino como se detalla en la figura 10.10; mientras que el mensaje enviado por el usuario desde el monitor serial del ambiente Arduino se recibe en la línea 10, para ser presentado sobre la ventana de comandos

(línea 11). Finalmente, la comunicación se desconecta en la línea 12, así como el área de memoria asignada al dispositivo remoto es liberada.



A Código MATLAB 10.3 cap10_Bluetooth_Matlab_B.m

Arduino. Aplicaciones en Robótica y Mecatrónica.

Capítulo 10 Bluetooth.

Fernando Reyes Cortés y Jaime Cid Monjaraz.

Alfaomega Grupo Editor: "Te acerca al conocimiento".

Archivo cap10_Bluetooth_Matlab_B.m

Versión de Matlab 2014a

- 1 clc
- 2 clear all
- 3 close all
- 4 format short
- 5 instrhwinfo('Bluetooth', 'HC-05') %información técnica del dispositivo remoto.
- 6 dispositivo_Bluetooth = Bluetooth('HC-05',1)%construye objeto clase Bluetooth.
- 7 fopen(dispositivo_Bluetooth) %abre comunicación Bluetooth: JY-MCU y MATLAB.
- $8 \ \ \text{mensaje} = \text{'Desde MatLab se transmite hacia el módulo JY-MCU conectado a la tarjeta Arduino UNO'} \ ;$
- 9 fwrite(dispositivo_Bluetooth, mensaje); %envía mensaje a la tarjeta Arduino.
- 10 cadena=fscanf(dispositivo_Bluetooth, '%c'); %lee el puerto serial Bluetooth.
- 11 cadena %despliega mensaje desde Arduino en la ventana de comandos MATLAB.
- 12 fclose (dispositivo_Bluetooth); %desconecta dispositivo remoto Bluetooth.
- 13 clear dispositivo_Bluetooth; %libera memoria asociada a dispositivo remoto.

Durante la ejecución del script cap10_Bluetooth_Matlab_B puede verificar en el panel de Dispositivos que el estatus del dispositivo remoto HC-05 se encuentra conectado, este estado se logra con la función fopen(...).



La figura 10.10 muestra la ventana del monitor serial Arduino donde se escribe el mensaje de texto a transmitir, así como la cadena de caracteres transmitida desde MATLAB y recibido por el módulo de comunicación inalámbrica Bluetooth JY-MCU. El mensaje enviado desde Arduino aparece en la ventana de comandos de MATLAB de la siguiente forma: fx >>Enviando mensaje desde Arduino.

Figura 10.10 Bluetooth.

A continuación se describe el procedimiento para llevar a cabo la aplicación de transmisión/recepción de mensajes entre MATLAB y la tarjeta Arduino UNO con módulo JY-MCU.

- Editar el sketch cap10_Bluetooth_B del cuadro de código Arduino 10.2.
- Compilar el sketch cap10_Bluetooth_B mediante el icono .
- Descargar el código de máquina a la tarjeta Arduino usando 👈
- Desde el ambiente de programación en MATLAB editar el código del cuadro de código en MATLAB 10.3 del script: cap10_Bluetooth_Matlab_B.
- Abrir la ventana del monitor serial usando el icono 🔑.
- Escribir el texto: Enviando mensaje desde Arduino \leftarrow .
- En el ambiente de programación en MATLAB ejecutar el programa cap10_Bluetooth_B.

El orden de pasos que se establece en este procedimiento es importante para una adecuación realización de la aplicación de comunicación serial por Bluetooth.

♣ ♣ ♣ Ejemplo 10.5

Considere el siguiente sistema dinámico de primer orden:

$$\dot{x}(t) = -4x(t) + 4u(t)$$
 (10.1a)

$$y(t) = x(t) (10.1b)$$

donde la señal de entrada $u(t) \in \mathbb{R}$ es un escalón unitario, $y(t) \in \mathbb{R}$ es la respuesta del sistema, $x(t) \in \mathbb{R}$ es la variable de estado. Implementar en lenguaje C el algoritmo de la solución discreta $x(t_k)$ del sistema dinámico (10.1a) y ejecutarlo sobre la tarjeta Arduino UNO; enviar la información de $x(t_k)$ por Bluetooth a MATLAB para su representación gráfica. Asimismo, verificar por simulación en MATLAB usando la función ode45(...) (integración numérica de Runge-Kutta) que la solución numérica x(t) corresponde a la respuesta $x(t_k)$ enviada por el Sistema Arduino.

Solución

El análisis, estudio y diseño de algoritmos de control bajo el enfoque modelo dinámico resulta una herramienta clave en ingeniería mecatrónica y robótica. En este contexto, los sistemas discretos con aplicación directa en procesamiento digital de señales y en control digital representan un área importante para el desarrollo de aplicaciones dentro de la automatización. Por tal motivo, el presente ejemplo tiene la finalidad de ilustrar una técnica de sistemas discretos y sus detalles claves para llevar a cabo la implementación práctica transmitiendo los resultados por Bluetooth.

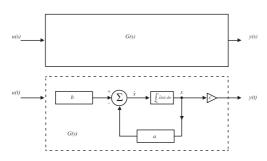


Figura 10.11 Sistema dinámico 10.1a.

Considere el caso general del sistema dinámico (10.1a) descrito por su diagrama a bloques de la figura 10.11. La parte superior de esta figura representa el diagrama de bloques correspondiente a la función de transferencia descrita por: $G(s) = \frac{y(s)}{u(s)} = c \frac{b}{s+a}$, donde $a, b, c \in \mathbb{R}_+$. Sin embargo, esta representación no ofrece ventajas sobre la evolución dinámica del sistema.

La representación en diagramas a bloques del modelo dinámico expresado en variables fase (parte inferior de la figura 10.11) permite obtener información interna sobre los cambios de estados x(t) en la dinámica del sistema (10.1a).

Con la finalidad de evitar aspectos empíricos, prefiriendo en su lugar formalidad académica sobre el desarrollo práctico, a continuación se presenta el análisis matemático que fundamenta la solución discreta $x(t_k)$ del sistema (10.1a).

La ecuación diferencial ordinaria de primer orden (10.1a) puede ser expresada de manera general como:

$$\dot{x}(t) = -ax(t) + bu(t) \tag{10.2}$$

donde $a,b \in \mathbb{R}_+$. Para obtener la solución x(t) en tiempo continuo se procede en detalle como a continuación se indica:

$$\dot{x}(t) + ax(t) = bu(t) (10.3a)$$

$$e^{at} [\dot{x}(t) + ax(t)] = e^{at}bu(t) (10.3b)$$

$$e^{at} \dot{x}(t) + e^{at}ax(t) = e^{at}bu(t) (10.3c)$$

$$\underbrace{e^{at} \dot{x}(t) + e^{at}ax(t)}_{dt} = e^{at}bu(t) (10.3d)$$

$$\frac{d}{dt} \left[e^{at} x(t) \right] = e^{at} b u(t) \quad (10.3e)$$

$$d \left[e^{at} x(t) \right] = dt e^{at} b u(t)$$

$$\int_{t_0}^t d \left[e^{a\sigma} x(\sigma) \right] = \int_{t_0}^t e^{a\sigma} b u(\sigma) d\sigma$$

$$e^{at} x(t) \Big|_{t_0}^t = \int_{t_0}^t e^{a\sigma} b u(\sigma) d\sigma$$

Por lo tanto, la solución en tiempo continuo x(t) de la ecuación (10.2) obtiene la siguiente forma:

$$x(t) = e^{-a(t-t_0)}x(t_0) + \int_{t_0}^t e^{-a(t-\sigma)}bu(\sigma)d\sigma \quad (10.4)$$

Para obtener la versión discreta, se muestrea la solución en tiempo continuo x(t) usando un determinado periodo de muestreo h y un retenedor de orden cero. El tiempo discreto se define como $t_k = t_{k-1} + h$, es decir t_k evoluciona como múltiplo de h.

Muestreando la solución x(t) (10.4) se obtiene $x(t_k)$:

$$x(t_k) = e^{-a(t_k - t_{k-1})} x(t_{k-1}) + \int_{t_{k-1}}^{t_k} e^{-a(t_k - \sigma_{k-1})} bu(\sigma_{k-1}) d\sigma_{k-1}$$
(10.5)

Note que: $t_k - t_{k-1} = h$, por lo que los límites de integración de t_{k-1} a t_k equivale a tomarlos de 0 a h; además, tomando en cuenta que mediante un retenedor de orden cero la señal $u(t_{k-1})$ es constante en el intervalo de integración, se tiene la siguiente ecuación:

$$x(t_k) = e^{-ah}x(t_{k-1}) + \int_0^h e^{-ah}bdh \ u(t_{k-1})$$
 (10.6)

La componente integral de la ecuación (10.6) se puede simplificar de la siguiente forma:

$$\int_0^h e^{-a\sigma}bd\sigma = \frac{b}{a} \left[1 - e^{-ah}\right]$$
 (10.7)

Finalmente, la versión recursiva $x(t_k)$ del sistema dinámico (10.2) que permite realizar aplicaciones en control digital está dada por:

364 Bluetooth

$$x(t_k) = e^{-ah}x(t_{k-1}) + \frac{b}{a} [1 - e^{-ah}] u(t_{k-1})$$
(10.8a)

$$y(t_k) = c x(t_k) (10.8b)$$

Desarrollo de la programación Bluetooth Arduino + MATLAB

El cuadro de código Arduino 10.4 contiene la programación en lenguaje C del algoritmo (10.8a) en el sketch **cap10_Bluetooth_C**, en el header se encuentran las librerías correspondientes para funciones matemáticas y comunicación serial por software: líneas 1 y 2, respectivamente. En la línea 3 se construye el objeto remoto tipo software serial, indicando que el pin 2 de la tarjeta Arduino tiene la función de recepción Rx, en este pin se conecta el pin Tx del módulo inalámbrico JY-MCU, mientras que el pin 3 tiene la función de transmisor Tx, el cual se conecta al pin Rx del módulo Bluetooth.

En las líneas 4 a 8 se encuentra la declaración de las variables utilizadas por el algoritmo. Sobre la línea 9 se encuentra la subrutina de configuración **setup()** donde se programa las velocidades de transmisión/recepción serial para el puerto USB y por software serial para comunicación con el módulo Bluetooth. A partir de la línea 13 se encuentra la subrutina principal de programación **loop()** del sketch **cap10_Bluetooth_C**.

En las líneas 14 a la 28 se encuentra el código de sincronización con el programa MATLAB cap10_Bluetooth_Matlab_C; inicialmente el semáforo bandera tiene un valor verdadero, haciendo que la secuencia del programa entre a un ciclo infinito $do\{...\}$ while(1); cuando MATLAB transmite en forma inalámbrica, en la línea 16 se detecta el enlace y en la línea 17 se registra la información enviada por MATLAB, se verifica que corresponde al comando de sincronía '95' en la línea 18, si esta información es verdadera, entonces la tarjeta Arduino envía al módulo JY-MCU la respuesta a través del comando '135' para iniciar la ejecución del algoritmo discreto $x(t_k)$ y el envío de datos por Bluetooth a MATLAB. Observe que el semáforo bandera es falso en la línea 19, esto hará que en subsecuentes eventos no entre a esta sección de código.

El proceso de ejecución del algoritmo para $x(t_k)$ se realiza para 10 segundos, con un periodo de muestreo h = 0.001 segundos, es decir: $t_k \in [0, h, 2h, 3h, \dots, 10]$; las condiciones iniciales para $t_0, u(0), x(0)$) son cero, en la línea 32 se realiza la evolución del tiempo discreto como múltiplos del periodo de muestreo h.

La señal de entrada escalón u(t) = 1 se lleva a cabo sobre la línea 43 y el cómputo de la solución discreta $x(t_k)$ (10.8a) se procesa en la línea 45. La transmisión de información por Bluetooth hacia

MATLAB de las variables tiempo discreto t_k y $x(t_k)$ se lleva a cabo a partir de la línea 50. Este proceso se realiza para diez mil puntos, es decir sobre el intervalo $t_k \in [0, h, 2h, 3h, \cdots, 10]$ segundos. La figura 10.12 muestra el perfil de la variable de estado $x(t_k)$ en el tiempo discreto, corresponde a diez mil puntos transmitidos por Bluetooth desde el módulo JY-MCU hacia **MATLAB** para su representación gráfica.

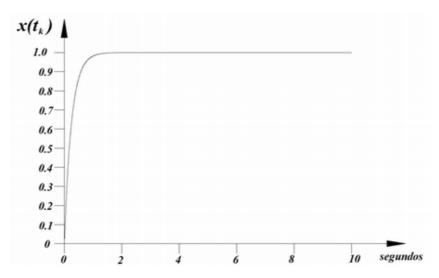


Figura 10.12 Respuesta x_{tk} en el Sistema Arduino (transmisión por Bluetooth).



Comunicación inalámbrica

Ejemplos adicionales de comunicación Bluetooth se encuentran disponibles en el sitio Web de esta obra; una serie de sketchs con la implementación práctica de sistemas discretos, algoritmos de control, adquisición de datos, señales analógicas y sensores, cuyos resultados se transmiten inalámbricamente para su representación gráfica en el ambiente de MATLAB.

366 Bluetooth

```
⊙ Código Arduino 10.4: sketch cap10_Bluetooth_C
  Arduino. Aplicaciones en Robótica y Mecatrónica.
  Capítulo 10 Bluetooth.
  Fernando Reves Cortés y Jaime Cid Monjaraz.
  Alfaomega Grupo Editor: "Te acerca al conocimiento".
   Sketch cap10_Bluetooth_C.ino
 1 #include <math.h> //librería para funciones matemáticas.
 2 #include <SoftwareSerial.h> //librería de software serial.
 {\bf 3} Software
Serial canal<br/>Bluetooth(2,3); //pin 2 es Rx, pin 3 es Tx por Software Serial.
4 float utk, tk, xtk; //variables que registran valores actuales.
 5 float tka=0, utka, xtka; //variables que registran los valores anteriores.
 6 float h=0.001; //periodo de muestreo.
 7 float a=4, b=4; //parámetros del sistema dinámico.
8 int bandera=1, comando=0;
9 void setup() {//subrutina de configuración.
10
       Serial.begin(9600); //velocidad serial por cable USB: pins 0 y 1.
       canalBluetooth.begin(9600); //velocidad serial por Software: pins 2 y 3.
11
12 }
13 void loop() {//lazo principal del sketch.
       if(bandera){ //código de sincronía con el programa cap10_Bluetooth_Matlab_C.m
14
         do{
15
16
             if(canalBluetooth.available()>0){ //¿Dato disponible en el módulo remoto?
                comando=canalBluetooth.read(); //lee comando enviado por MATLAB.
17
             if (comando==95){ ¿Comando correcto de sincronía?
18
                bandera=0; //se limpia el semáforo.
19
                canalBluetooth.print(135.0000,4); //envía comando de respuesta.
20
                canalBluetooth.print(" " ); //separación de datos.
                canalBluetooth.println(135.0000,4); //copia comando de respuesta.
22
                break;
23
             }
24
             delay(300);
25
26
          \}while(1);
27
```

Continúa código Arduino 10.4a: sketch cap10_Bluetooth_C

Arduino. Aplicaciones en Robótica y Mecatrónica.

Capítulo 10 Bluetooth.

Fernando Reyes Cortés y Jaime Cid Monjaraz.

Alfaomega Grupo Editor: "Te acerca al conocimiento".

Continuación del sketch cap10_Bluetooth_C.ino

```
29
        if(tk==10){//se repite el proceso para el intervalo t_k \in [0, h, 2h, ...10].
          tka=0; condición inicial: t_0 = 0, cuando k = 1.
30
31
32
        tk=tka+h; //tiempo discreto: t_k = kh, evoluciona como: t_k = t_{k-1} + h.
33
        /* Solución discreta del sistema dinámico (10.2).*/
34
        if(tk==h){//condiciones iniciales de las variables de control.
35
          xtka=0; //estado anterior x(t_{k-1}) = x(0) = 0, cuando k = 1.
36
          utka=0; //entrada al sistema: u(t_{k-1}) = u(0) = 0, cuando k = 1.
37
38
          canalBluetooth.print(tk-h,4); //transmite las condiciones iniciales.
39
          canalBluetooth.print(""); //espacio en blanco.
          canalBluetooth.println(xtka,4); //condición inicial x(0).
40
41
        else
42
          utk=1; //entrada de un escalón: u(t_k) = 1 si t_k > 0, u(0) = 0.
43
        /* x(t_k) = e^{-ah}x(t_{k-1}) + \frac{b}{a} [1 - e^{-ah}] u(t_{k-1})*/
44
        xtk = exp(-a*h)*xtka + (b/a)*(1-exp(-a*h))*utka;
45
        utka=utk; //actualiza entrada anterior: u(t_{k-1}) = u(t_k).
46
        xtka=xtk; //actualiza estado anterior: x(t_{k-1}) = x(t_k).
47
48
        //...
        /*Sección de transmisión de datos de la tarjeta Arduino a MATLAB.*/
49
        canalBluetooth.print(tk,4); //se transmite t_k, con cuatro fracciones.
50
        canalBluetooth.print(" " ); //espacio en blanco, para separar datos.
51
        canalBluetooth.println(xtk,4); //variable de estado x(t_k), con cuatro fracciones.
52
        //...
53
        tka=tk; //t_{k-1} = t_k.
54
        delay(1);//retardo por h.
55
56 }
```

368 Bluetooth

```
A Código MATLAB 10.5 cap10_Bluetooth_Matlab_C.m
  Arduino. Aplicaciones en Robótica y Mecatrónica.
  Capítulo 10 Bluetooth.
  Fernando Reyes Cortés y Jaime Cid Monjaraz.
  Alfaomega Grupo Editor: "Te acerca al conocimiento".
   Archivo cap10_Bluetooth_Matlab_C.m
                                                         Versión de Matlab 2014a
 1 clc clear all close all format short %libera variables olvidadas de memoria.
 2 disp('Buscando dispositivo Bluetooth.')
 3 disp('Momento por favor, este proceso puede durar varios segundos.')
 4 instrhwinfo('Bluetooth', 'HC-05');
 5 dispositivo_Bluetooth = Bluetooth ('HC-05',1);
 6 disp('Enlazando con dispositivo Bluetooth.')
 7 fopen(dispositivo_Bluetooth); %abre conexión remota con el dispositivo Bluetooth.
 8 disp('Dispositivo Bluetooth conectado: momento por favor...');
 9 xlabel ('Segundos'); ylabel ('Datos'); title ('Adquisición de datos Arduino');
10 grid on; hold on;
11 prop = line(nan,nan, 'Color', 'b', 'LineWidth', 1); %tipos de líneas a graficar.
13 disp('Sincronizando comunicación...');
14 fscanf(dispositivo_Bluetooth, '%f %f',[2,1])%limpiando basura residual serial.
15 disp('Iniciando sincronía con el dispositivo remoto.');
16 while (1)
17
       fwrite(dispositivo_Bluetooth, 95); %envía comando de sincronización.
18
       informe=fscanf(dispositivo_Bluetooth, '%f
                                                   \%f',[2,1]); %lee puerto Bluetooth.
       if (informe(1,1)==135.0000) % verifica comando de sincronización de Arduino.
19
          break;
       end
21
       pause on
22
23
       pause(0.3)
24
       pause off
26 disp('Adquisición de datos en proceso')
```

27 i=1; %valor inicial del pivote de registro de datos.

Continúa código MATLAB 10.5a: cap10_Bluetooth_Matlab_C.m

Arduino. Aplicaciones en Robótica y Mecatrónica.

Capítulo 10 Bluetooth.

Fernando Reyes Cortés y Jaime Cid Monjaraz.

Alfaomega Grupo Editor: "Te acerca al conocimiento".

Continuación de cap10_Bluetooth_Matlab_C.m

```
28 while i<10000
       % adquisición de diez mil datos del Sistema Arduino por Bluetooth.
       datos = fscanf(dispositivo_Bluetooth, '%f %f ,[2, 1]); %lee puerto Bluetooth.
30
       tiempo(i)=datos(1,1);// registro del tiempo discreto t_k.
31
       xtk(i)=datos(2,1);// variable de estado x(t_k).
32
33
       set(prop, 'YData', xtk(1:i), 'XData', tiempo(1:i));
       drawnow;//presentación gráfica en el tiempo de las variable de estado x(t_k) vs t_k.
34
      i=i+1;//incrementa pivote de registros.
35
36 end
37 %
38 % Simulación del sistema dinámico \dot{x}(t) = -ax(t) + bu(t), usando Runge-Kutta 4/5.
39 ti=0; %tiempo inicial de simulación (segundos).
40 tf=10; %tiempo de final (segundos).
41 h=0.001; %periodo de muestreo (segundos).
42 ts=ti:h:tf; %vector de tiempo.
43 condicion_inicial=0; \%x(0) = 0.
44 opciones=odeset('RelTol', 1e-6, 'InitialStep', h, 'MaxStep', h);
45 %La simulación por Runge-Kutta igual contempla diez mil puntos, h = 0.001.
46 % Solución numérica Runge Kutta 4/5 x(t) del sistema (10.2). %
47 [t, x]=ode45('cap10_sl', ts, condicion_inicial, opciones); %Runge Kutta 4/5.
48 %
49 figure %genera una ventana diferente de drawnow.
50 %Gráfica comparativa de x(t) vs x(t_k); los perfiles de ambas gráficas se superponen.
51 plot(tiempo,xtk, t, x) % compara soluciones numéricas x(t) vs versión recursiva x(t_k).
52 %
53 fclose (dispositivo_Bluetooth); %desconecta dispositivo remoto Bluetooth.
```

54 clear dispositivo_Bluetooth; %libera espacio de memoria asignado a Bluetooth.

370 Bluetooth

Con respecto al programa en MATLAB cap10_Bluetooth_Matlab_C que realiza el enlace en Bluetooth con el módulo inalámbrico JY-MCU se encuentra documentado en el cuadro de código MATLAB 10.5; la construcción del objeto Bluetooth HC-05 se lleva a cabo en la línea 5, cuya conexión se establece tal y como se ilustra en la línea 7. Antes de iniciar el proceso de adquisición de datos, se limpia de posible basura residual de datos el buffer serial (ver línea 14). Posteriormente, de las líneas 16 a la 21, se establece la fase de sincronización con la tarjeta Arduino, enviando el comando '95' y esperando respuesta '135.000'. Esta sección de código está directamente relacionada con las líneas 14 a la 28 del sketch cap10_Bluetooth_C descrito en el cuadro de código Arduino 10.4.

La adquisición de datos que envía el módulo remoto Bluetooth desde la tarjeta Arduino se registra en las variables **tiempo** y **xtk** para su presentación gráfica (ver líneas 28 a la 36). El graficado de estos puntos se realiza punto a punto conforme llegan usando **drawnow**, generando la figura 10.12. Para propósitos de comparar la eficiencia del algoritmo recursivo $x(t_k)$ ecuación (10.8a) que se ejecuta sobre la tarjeta Arduino UNO, se realiza en **MATLAB** la integración numérica por Runge Kutta 4/5 **ode45(...)** (línea 47) del modelo dinámico (10.1a), el cual se encuentra implementado en el cuadro de código **MATLAB** 10.6, script **cap10_sl**).

```
A Código MATLAB 10.6 cap10_sl.m
 Arduino. Aplicaciones en Robótica y Mecatrónica.
 Capítulo 10 Bluetooth.
 Fernando Reyes Cortés y Jaime Cid Monjaraz.
 Alfaomega Grupo Editor: "Te acerca al conocimiento".
  Archivo cap10_sl.m
                                                       Versión de Matlab 2014a
1 function xp=cap10_sl(t,x)
2
      %Este archivo deberá estar en la misma carpeta de cap10_Bluetooth_Matlab_C.m
3
      a=4; %parámetro del sistema dinámico (ancho de banda).
     b=4; %parámetro del sistema dinámico.
4
     u(t==0)=0; % señal de entrada escalón u(0)=0.
5
     u(t>0)=+1; % si t>0, u(t)=1.
6
     xp=-a*x+b*u; \%\dot{x}(t) = -ax(t) + bu(t).
7
8 end
```

El proceso de simulación del modelo dinámico (10.1a) se realiza para el mismo intervalo de tiempo $t \in [0, h, 2h, 3h, \dots, 10]$ segundos y para el mismo paso de integración h, es decir se procesan diez mil puntos. El lector puede comprobar que las soluciones obtenidas por Runge-Kutta x(t) y la versión discreta usando un retenedor de orden cero $x(t_k)$ implementada en Arduino coinciden, esto se verifica graficándolas simultáneamente en la misma escala como se indica en la línea 51 del cuadro

de código MATLAB 10.5a. Finalmente, la conexión Bluetooth se cierra en la línea 53, liberando la memoria asignada al dispositivo remoto en la línea 54.

Observaciones importantes en la comunicación de datos

Los datos que se envían y reciben en comunicación serie son por medio de paquetes de mensajes de texto, por lo tanto es clave indicar cómo se separan los datos entre sí dentro de esos paquetes y dónde termina el paquete; de no ser así, existirán errores en la conversión de la información.

La manera de separar datos dentro de un mensaje es insertando al menos un espacio en blanco entre cada elemento del paquete, por ejemplo, con **canalBluetooth.print("")** y el último dato a transmitir se indica la terminación del paquete con retorno de carro, seguido de una nueva línea de texto usando: **canalBluetooth.println(dato)**.

Si no se inserta un espacio en blanco entre los datos a transmitir, en la recepción aparecerán unidos dichos datos. Considere el escenario siguiente:

```
canalBluetooth.print(135);//transmite el primer dato.
canalBluetooth.println(156);//transmite segundo dato con fin de paquete.
```

En MATLAB al momento de recepción de datos se realiza con la función:

```
\mathbf{x} = \mathbf{fscanf}(\mathbf{dispositivo\_Bluetooth}, '\%\mathbf{li} \%\mathbf{li'}, [2, 1]); //\mathbf{donde} \mathbf{x} \in \mathbb{R}^{2 \times 1}.
```

La recepción equivale a una sola cadena de caracteres: 135156; para este caso, el paquete está compuesto de dos elementos enteros largos (long) que son registrados en el vector \mathbf{x} (la dimensión de \mathbf{x} se indica con el último argumento [2, 1] de la función fscanf(...)).

Sin embargo, la conversión es incompleta, ya que el primer elemento de \mathbf{x} se le asigna: $\mathbf{x}(\mathbf{1},\mathbf{1})=135156$, mientras que para el segundo elemento $\mathbf{x}(\mathbf{2},\mathbf{1})$ queda inconcluso.

El código correcto para separar e indicar fin de paquete es el siguiente:

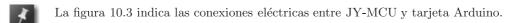
```
canal
Bluetooth.print(135);//transmite el primer dato. canal
Bluetooth.print(" " );//indica seperación de datos. canal
Bluetooth.println(156);//envía segundo dato con fin de paquete.
```

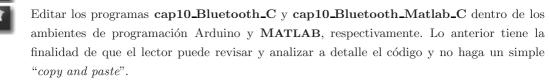
El resultado correcto es: $\mathbf{x}(1,1)=125 \ \mathbf{y} \ \mathbf{x}(2,1)=156$.

372 Bluetooth

Procedimiento para ejecutar correctamente la aplicación Bluetooth

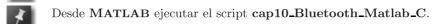
Con la finalidad de que el lector no tenga problemas para ejecutar la aplicación de comunicación Bluetooth entre los programas cap10_Bluetooth_C y cap10_Bluetooth_Matlab_C se describen los pasos en secuencia cronológica; resaltando llevarlos a cabo tal y como se establecen a continuación:













Ejemplos ilustrativos

Ejemplos de adquisición de datos y procesamiento digital de señales con aplicaciones en Bluetooth se encuentran disponibles en el sitio Web de esta obra.



Aplicaciones de control

En el sitio Web de este libro, puede descargar ejemplos de control de sistemas discretos lineales y no lineales con comunicación Bluetooth.



10.5 Resumen

B LUETOOTH es una tecnología abierta para comunicación inalámbrica que intercambia información en ambos sentidos (transmisión/recepción) en distancias cortas a través de señales

Alfaomega

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

Fernando Reyes Cortés • Jaime Cid Monjaraz

10.5 Resumen 373

de radio de longitud corta desde dispositivos fijos y móviles usando un protocolo de comunicación basado en paquetes.

Esta tecnología ofrece seguridad y flexibilidad en la comunicación para conectar e intercambiar información entre dispositivos que soporten el protocolo de comunicación SPP tales como robots manipuladores, adaptadores USB, sensores inalámbricos, tablets y teléfonos celulares y fijos con Sistema Android, laptops con Sistema Operativo Windows, impresoras, GPS, etc.

El Sistema Arduino incluye un conjunto de funciones para realizar interface con dispositivos remotos para comunicación serial inalámbrica dentro de la librería **SoftwareSerial**; por medio de este soporte es posible llevar a cabo enlace e intercambio de datos (transmisión y recepción) con el módulo remoto JY-MCU.

El número de aplicaciones en el ámbito de la ingeniería mecatrónica y robótica se incrementan mediante comunicación inalámbrica Bluetooth, ya que no requiere cableado o conectar subsistemas digitales como parte del sistema empotrado, facilitando el diseño y versatilidad a bajo costo y bajo consumo de potencia.

Por otro lado, el ambiente de programación de MATLAB contiene una herramienta específica para el desarrollo de aplicaciones con Bluetooth: The Instrument Control ToolboxTM Bluetooth, permitiendo enlace de comunicación bidireccional con la tarjeta Arduino y el módulo inalámbrico JY-MCU.

Combinando las características de las funciones de la librería **SoftwareSerial** con **The Instrument Control Toolbox**TM **Bluetooth** de **MATLAB** se puede prescindir de programas comerciales como los utilizados en los ejemplos 10.1 al 10.3, ya que no proporcionan todos los requerimientos necesarios que se utilizan en la implementación práctica de algoritmos de control y en la automatización de procesos específicos.

Es decir, aquellas aplicaciones comerciales son de arquitectura cerrada, por lo que no resuelven los requerimientos del proceso a automatizar. De esta forma, el usuario a través de **MATLAB** cuenta con herramientas apropiadas para diseñar aplicaciones cubriendo cualquier requerimiento en la automatización de procesos.

374 Bluetooth



10.6 Referencias selectas

B LUETOOTH como tecnología estratégica contiene una diversidad de bibliografía y documentos técnicos de consulta, sin embargo, con la finalidad de que el lector profundice a detalle en este tipo de comunicación inalámbrica se recomienda ampliamente revisar las siguientes referencias:

Especificaciones técnicas sobre el estándar de comunicación Bluetooth proporcionadas por Grupo de Interés Especial de Bluetooth (SIG) se pueden encontrar en:



https://www.bluetooth.org/Technical/Specifications/adopted.htm



"IEEE Std 802.15.1-2002 -IEEE Standar for Information Technology-Telecommunications and information exchange between systems- Local and metropolitan area networks- Specifications requirements Part 15.1: Wireless Medium Access Control (MAC) and Physical Layer (PHY) pecifications for Wireless Personal Area Nteworks (WPANs)". doi:10.1109/IEEESTD.2002.93621

La versión mejorada de **SoftwareSerial** que corresponde a versiones posteriores a 1.0 del Sistema Arduino se basa en la librería **NewSoftserial** desarrollada por Mikal Hart, cuya información se encuentra disponible en el siguiente enlace:



http://www.arduiniana.org

Para manejo simultáneo de flujo de datos en diferentes puertos digitales de las tarjetas Arduino se puede utilizar la librería **AltSoftSerial** de Stoffregen:



http://pjrc.com

Tiendas electrónicas donde puede encontrar el módulo remoto JY-MCU y otros dispositivos para interface son las siguientes:



http://www.core-electronics.com.au



http://www.digikey.com/



https://www.jameco.com/



http://www.teslabem.com

La interface **Hercules** es un software de uso libre para comunicación Bluetooth, diseñado por HWgroup, se puede descargar del siguiente enlace:



http://www.hw-group.com/

Desarrollo de métodos numéricos en lenguajes C y C++ pueden obtenerse en:



William H Press, Saul A. Teukoisky, William T. Vetterling, & Brian P. Flannery. "Numerical Recipes in C: The Art of Scientific Computing". Cambridge University Press. 1992.



William H Press, Saul A. Teukoisky, William T. Vetterling, & Brian P. Flannery. "Numerical Recipes in C++: The Art of Scientific Computing". Cambridge University Press. Second Edition. 2002.

Lenguaje MATLAB y sus aplicaciones en robótica y mecatrónica se describen en:



Fernando Reyes Cortés. "MATLAB Aplicado a Robótica y Mecatrónica". Alfaomega Grupo Editor. 2012.

Fundamentos de sistemas discretos y control de procesos por computadora se encuentran en:



Karl J. Åström & Björn Wittenmark. "Computer Controlled Systems: Theory and Design". Prentice-Hall, Editor Thomas Kailath, Third Edition. 1997.

10.7 Problemas propuestos



H oy en día, comunicación inalámbrica por Bluetooth es una herramienta importante de la tecnología moderna, particularmente tiene enorme impacto en la transmisión y recepción de información en robótica, mecatrónica y en general en todas las áreas de la ingeniería y ciencias exactas.

A continuación se presenta un conjunto de problemas para valorar los conocimientos adquiridos en este capítulo.

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

Fernando Reyes Cortés • Jaime Cid Monjaraz

376 Bluetooth

10.7.1 Diseñar un sketch que se ejecute en algún modelo de tarjeta Arduino, para transmitir a través de comunicación Bluetooth hacia un teléfono celular con Sistema Android, los números pares comprendidos entre 0 y 20.

- 10.7.2 Realizar un algoritmo para transmitir inalámbricamente los números impares del intervalo [0, 100] desde un módulo remoto conectado a una tarjeta Arduino hacia un dispositivo tablet.
- 10.7.3 Considere un dispositivo tablet con Sistema Android y el módulo remoto JY-MCU conectado a una tarjeta Arduino. Desde el dispositivo tablet enviar al módulo remoto comandos usando los siguientes caracteres: 'a', 'b', 'c', 'd' y 'e' tal que en la tarjeta Arduino envíe como respuesta a la tablet: "Tarea A ejecutándose.", "Tarea B ejecutándose.", "Tarea C ejecutándose.", "Tarea D ejecutándose." y "Tarea E ejecutándose.", respectivamente. Sugerencia: en la programación en lenguaje C asociada a la tarjeta Arduino utilice la instrucción switch(){... case...}.
- 10.7.4 En relación con el ejercicio 10.5, considere el sistema dinámico 10.1a. En lugar de utilizar la señal escalón u(t), realice la ejecución del algoritmo discreto $x(t_k)$ tomando en cuenta los siguientes casos de entrada:

```
a) u(t) = \alpha \tanh(1-x).
```

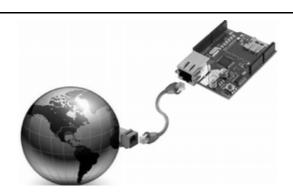
- b) $u(t) = \alpha (1.5 x)$.
- c) $u(t) = \alpha \operatorname{senh}(2 x)$.
- d) $u(t) = \alpha \arctan (10 x)$.

donde $\alpha \in \mathbb{R}_+$ es una constante a sintonizar en cada caso por el usuario. Compruebe por simulación en **MATLAB** que la solución x(t) coincide con $x(t_k)$.

- 10.7.5 Diseñe un sketch para que pueda enviar la información de la siguientes funciones a **MATLAB** usando Bluetooth:
 - a) sen(t).
 - b) $\cos(t)$.
 - c) $1 e^{-0.02t}$.
 - d) $\tanh(t)$.
 - e) atan((t)).
 - f) $\cosh(t)$.
 - g) senh(t).

donde $t \in \mathbb{R}_+$ es la variable que representa la evolución del tiempo. En MATLAB grafique la respuesta de cada función.

Capítulo



- 11.1 Introducción
- 11.2 Tecnología de Ethernet
- 11.3 Trama de Ethernet
- 11.4 Arduino Ethernet Shield
- 11.5 Librería Ethernet
- 11.6 Ejemplos prácticos
- 11.7 Resumen
- 11.8 Referencias selectas
- 11.9 Problemas propuestos

Competencias

Presentar la programación que permite la configuración de la tarjeta Ethernet Shield al protocolo de comunicación Web para expandir las aplicaciones que tiene el sistema Arduino en el ámbito global. Se describen ejemplos prácticos que ilustran la forma de desplegar información en una página Web.

Desarrollar habilidades en:

- Programación de aplicaciones Web usando las funciones de la librería Ethernet.
- Descripción de la tecnología Ethernet.
- Asignación de direcciones IP para desplegar información de variables y procesos de automatización.
- Características de programación para cliente/servidor.
- Adquisición de datos, cálculos numéricos y procesamiento de información.

11.1 Introducción



E thernet es un estándar de redes de área local para computadoras, define las características de cableado y señalización de nivel físico y los formatos de tramas de datos del nivel de enlace de datos para el modelo OSI (Open Systems Interconnection). La norma de Ethernet fue definida por el Instituto de Ingenieros Eléctricos y Electrónicos (IEEE Institute of Electrical and Electronics Engineers) como IEEE Standard 802.3. Actualmente, Ethernet es la tecnología que mantiene un balance entre velocidad de comunicación, costo y facilidad de instalación; además, cuenta con una amplia aceptación en el mercado y la habilidad de soportar virtualmente todos los protocolos de redes populares.

El nombre Ethernet proviene del antiguo concepto físico de "eter" un fluido que se suponía llenaba todo el espacio y las ondas electromagnéticas viajaban por este medio. Sin embargo, desde 1887 a través del experimento Michelson & Morley se demostró que el **éter** no existe. Como una analogía del "éter", el cable coaxial representa el medio por donde viaja la señal y la idea es que la red de computadoras se encuentre en todos lados.

En 1972 el Ethernet fue desarrollado en la compañía Xerox en PARC (Palo Alto Research Center) por Robert Metcalfe como un sistema de red denominado **Ethernet Experimental**, cuyo objetivo era conseguir un medio de comunicación entre computadoras. En una red de Ethernet, todos los equipos están conectados a la misma línea de comunicación (cable coaxial o fibra óptica) y la transmisión de información se lleva a cabo con acceso al medio por detección de la onda portadora y con detección de colisiones usando el protocolo CSMA/CD (Carrier Sense Multiple Access with Collision Detect).

Una red de área local LAN (Local Area Network) es aquella que se conectan redes de computadoras confinadas en un área geográfica, como puede ser un edificio o una ciudad y que pueden ser usadas por una gran cantidad de usuarios. Las redes de área extensa WAN (Wide Area Network) conectan múltiples redes LAN que están geográficamente dispersas. Esto se realiza conectando las diferentes LANs mediante servicios que incluyen líneas telefónicas alquiladas (punto a punto), líneas de teléfono normales con protocolos síncronos y asíncronos, tendido de cables coaxiales y por fibra óptica, enlaces vía satélite, y servicios portadores de paquetes de datos. Hoy en día la conectividad entre redes de computadoras es utilizada por millones de usuarios. Internet fue usado inicialmente por el ejército e instituciones de investigación científica; los sitos Word Wide Web (www) proporcionan recursos informáticos, educativos, políticos, economía y se encuentran prácticamente en cualquier lugar de una ciudad. Por otro lado, Intranet es una red privada que utiliza herramientas del tipo de internet, pero disponible sólo dentro de esa organización. Una Intranet permite un modo de acceso fácil a información corporativa para los empleados a través del mismo tipo de herramientas

que emplean para moverse fuera de la compañía.

11.2 Tecnología de Ethernet



H oy en día, Ethernet es tecnología indispensable en todas las actividades cotidianas, desde la década de los 90's llegó a posicionarse con gran aceptación a nivel internacional; actualmente, se han desarrollado adaptadores que permite enlazar la conexión con versiones anteriores y que manejan protocolos diferentes.

En la figura 11.1 se muestran las capas del modelo OSI. Los elementos que constituyen la capa física de Ethernet son de dos tipos: pasivos y activos; los dispositivos pasivos generan y/o modifican la señal (cables, jacks/conectores, patch panels), mientras que los activos son dispositivos que transmiten la señal como transceptores, repetidores (multipuertos hubs). Los protocolos que definen a Ethernet se ubican en la capa física y en la mitad inferior de la capa de enlace de datos, conocida como subcapa de control de acceso al medio MAC (Media Access Control).



Figura 11.1 Capas del modelo OSI.

En la capa de enlace de datos se realiza el direccionamiento local, detección de errores y control del acceso en la capa física, además también interviene la compatibilidad de tecnología y la comunicación con la computadora; aquí es donde interviene la subcapa MAC que se ocupa de los componentes físicos que se utilizan para comunicar la información y preparar los datos para transmitirlos a través de los medios. La subcapa de control de enlace lógico LLC (*Logical Link Control*) es independiente del equipo físico que interviene en el proceso de comunicación.

La arquitectura Ethernet es una red de conmutación de paquetes de acceso múltiple y difusión amplia que utiliza un medio pasivo y sin control central, proporcionando detección de errores, sin llegar a su corrección.

El acceso al medio de transmisión se rige desde las propias estaciones mediante arbitraje estadístico; los paquetes de datos transmitidos alcanzan a todas las estaciones, siendo cada estación responsable de reconocer la dirección contenida en cada paquete e identificar las direcciones propias. Ethernet realiza varias funciones que incluyen desde empaquetado y desempaquetado de los datagramas, manejo del enlace, codificación y decodificación de datos y acceso del canal.

El manejador del enlace es responsable de vigilar el mecanismo de colisiones, escuchando hasta que el medio de transmisión está libre antes de iniciar una transmisión, sólo un usuario utiliza la transmisión cada vez (banda base). El manejo de colisiones se realiza deteniendo la transmisión y esperando cierto tiempo antes de volverlo a intentar.



Los protocolos de red son normas que permiten a las computadoras comunicarse, los cuales definen la forma en que sus elementos deben identificarse entre sí, así como la forma en que los datos deben transitar por la red, y cómo esta información debe procesarse una vez que alcanza su destino final.

Los protocolos también definen procedimientos para gestionar transmisiones o paquetes perdidos o dañados. Dentro de los protocolos más utilizados en la actualidad se encuentran: IPX (Novell NetWare), TCP/IP (UNIX y Sistemas Operativos Windows), AppleTalk (Sistema Operativo Macintosh), y NetBIOS/NetBEUI (para redes LAN Manager y WindowsNT). A pesar de que cada protocolo es diferente, todos pueden compartir el mismo cableado físico, esto se le conoce como independencia de protocolos sobre el mismo medio físico.

En 1985 el Comité de estándares para las redes metropolitanas y locales de IEEE publicó los estándares para LAN, los cuales inician con el número 802, específicamente los estándares para Ethernet corresponde al 802.3, la versión modificada que incluye actualizaciones para garantizar compatibilidad corresponde a 802.3.

Actualmente Ethernet es la capa física más popular de la tecnología LAN. Fast Ethernet es otro tipo de tecnología que requiere mayor velocidad. Se estableció la norma Fast Ethernet (IEEE 802.3u), aumentando la velocidad de comunicación de 10 Megabits por segundo (Mbps.) a 100 Mbps, con cambios mínimos en la estructura del cableado existente. Hay tres tipos de Fast Ethernet: 100BASE-TX para el uso con cable UTP de categoría 5, 100BASE-FX para el uso con cable de fibra óptica, y 100BASE-T4 que utiliza un par de cables más para permitir el uso con cables UTP de categoría 3. La norma 100BASE-TX se ha convertido en la más popular debido a su compatibilidad con la norma Ethernet 10BASE-T.

En la tabla 11.2 se muestran las principales características de tecnología Ethernet que definen longitud de comunicación y velocidad de enlace para cable coaxial y fibra óptica.

 ${\bf Tabla\ 11.2\ Tecnología\ Ethernet}.$

| Tecnología | Velocidad de trans- misión | Tipo de cable Distancia | | Topología | |
|------------|-------------------------------|----------------------------------|--------|--|--|
| 10Base2 | 10 Mbps | Coaxial delgado | 185 m | Bus: conector T | |
| 10Base5 | 10 Mbps | Coaxial grueso | 500 m | Bus: conector T | |
| 10BaseT | 10 Mbps | Par trenzado | 100 m | Estrella: Hub, Switch | |
| 10BaseF | 10 Mbps | Fibra óptica | 2000 m | Estrella: Hub o Switch | |
| 100BaseTX | 100 Mbps | Par trenzado: categoría 5 UTP | 100 m | Estrella: half duplex (hub) y full duplex (switch) | |
| 1000BaseSX | 1000 Mbps | Fibra óptica (multimodo) | 550 m | Estrella: full duplex (switch) | |
| 1000BaseLX | 1000 Mbps | Fibra óptica (monomodo) | 5000 m | Estrella: full duplex (switch) | |



Topologías de redes

Las redes Ethernet se encuentran generalmente en dos topologías: bus y estrella; ambas definen cómo se conectan los nodos entre sí. Un nodo es un dispositivo activo conectado a la red, como una computadora o impresora, concentrador, conmutador, router, etc.

La topología de bus tiene los nodos en serie, con cada nodo conectado a un cable largo o bus. Varios nodos pueden conectarse en el bus y pueden empezar la comunicación con el resto de los nodos en ese segmento del cable. Una ruptura en cualquier parte del cable causará que el segmento entero deje de funcionar, hasta que se repare. Como ejemplos de topología de bus tenemos a 10BASE-2 y 10BASE-5.

Por otro lado, 10BASE-T Ethernet y Fast Ethernet conectan una red de computadoras mediante topología estrella. Generalmente una computadora se sitúa en un extremo del segmento y el otro extremo termina con un concentrador.

La principal ventaja de este tipo de red es la fiabilidad, dado que si uno de los segmentos punto a punto tiene una rotura, afectará sólo a los dos nodos en ese eslabón. Otros usuarios de la red continuarán operando como si ese segmento no existiera.

Los dispositivos que se utilizan en Ethernet son: tarjetas de red, repetidores, concentradores, puentes, conmutadores, nodos de red y el medio de interconexión. Los nodos de red pueden clasificarse en dos grandes grupos: equipo terminal de datos y equipo de comunicación de datos.

El equipo terminal de datos son dispositivos de red que generan el destino de los datos: computadoras, routers, estaciones de trabajo, servidores de archivos, servidores de impresión; todos son parte del grupo de las estaciones finales.

Por otro lado, el equipo de comunicación de datos son los dispositivos de red intermediarios que reciben y retransmiten las tramas dentro de la red; pueden ser: conmutadores (switch), concentradores (hub), repetidores o interfaces de comunicación.

Concentrados (hub) funcionan como sistemas repetidores con múltiples nodos de salida, reciben la señal por uno de sus puertos y la repiten por todos sus puertos de salida, permitiendo la conexión con varios nodos.

Puente (bridge) interconecta segmentos de red haciendo el cambio de frames (tramas) entre las redes de acuerdo con una tabla de direcciones que le dice en qué segmento está ubicada una dirección MAC dada. Se diseñan para uso entre LAN's que usan protocolos idénticos en la capa física y MAC (de acceso al medio). Aunque existen bridges más sofisticados que permiten la conversión de formatos MAC diferentes (Ethernet-Token Ring por ejemplo).

Conmutador (switch) funciona parecido al bridge, pero permite la interconexión de múltiples segmentos de red, funciona en velocidades más rápidas y es más sofisticado. Los switches pueden tener otras funcionalidades, como redes virtuales, y permiten su configuración a través de la propia red.

Para conectar nodos a los diversos medios físicos Ethernet se usan **transceptores**. La mayoría de las computadoras y tarjetas de interfaz de red incorporan en su electrónica un transceptor 10BASE-T o 10BASE2, permitiendo ser conectados directamente a Ethernet sin requerir un transceptor externo.

Otros dispositivos compatibles Ethernet más antiguos incorporan un conector AUI para permitir al usuario conectarlo a cualquier medio físico, a través de un transceptor externo. El conector AUI consta de un conector de tipo DB de 15 pines, hembra en el lado de la computadora, macho en el lado del transceptor. Los cables coaxiales gruesos (10BASE5) también usan transceptores para permitir las conexiones.

Para conectar una computadora a una red se emplean tarjetas de interfaz de red, normalmente llamadas NIC (Network Interface Card); proporcionan conexión física entre el cable de la red y el bus interno de la computadora. Cuando una computadora está conectada a la red, entonces se denomina nodo y cada tarjeta tiene una única dirección MAC que la identifica en la red.

Los **repetidores** son sistemas electrónicos que aumentan el alcance de la conexión física, recibiendo señales y las retransmite por el mismo medio para evitar su degradación; se utilizan para unir dos áreas locales de igual tecnología.

Un **concentrador** al igual que un repetidor, toma cualquier señal entrante y la repite hacia todos los puertos; si el concentrador se conecta al troncal, entonces todas las computadoras situadas al final de los segmentos del par trenzado pueden comunicarse con todos los servidores en el troncal.

11.3 Trama de Ethernet



La trama o datagrama de Ethernet es el encapsulado del paquete de datos que transmite o recibe a través del medio; estos datos incluyen varias secciones o campos de información definidos por el protocolo de Ethernet, cuyo estándar original definió el mínimo y máximo tamaño de la trama en 64 bytes y 1518 bytes, respectivamente. Cuando el tamaño de una trama transmitida se sale de los rangos mínimo o máximo, el dispositivo receptor descarta la trama, considerándola no válida; sin embargo, esto da origen a las colisiones o señales no deseadas.

| Estándar IEEE 802.3 | | | | | | | | |
|--|--------------------------------------|----------------------|---------------------|--|-----------------|-----------------------|--------------------------|--|
| 7 | 1 | 6 | 6 | | 2 | 46 a 1500 | 4 | |
| Preámbulo | Delimitador de inicio de trama | Dirección destino | Dirección origen | | Longitud y tipo | Encabezado y datos | Verificación de trama | |
| Preámbulo Este campo contiene 7 bytes | | | | Delimitador de inicio de trama Campo de un byte | | | | |
| Dirección destino Dirección MAC destino 6 bytes | | | | Dirección origen Dirección MAC origen 6 bytes | | | | |
| Longitud/Tipo Tipo de protocolo encapsulado Longitud de trama de 2 bytes | | | | Encabezado y Datos 802.2 Paquete encapsulado de datos de 46 a 1500 bytes, más relleno en caso de ser necesario | | | | |

Figura 11.2 Campos de trama en Ethernet.

La figura 11.2 muestra la estructura y tamaño en bytes de cada campo del datagrama. Incluye: preámbulo, campos de dirección de origen y destino, tipo de campo y datos, finaliza con la verificación de datos del frame.

Los campos de preámbulo y delimitador de inicio de trama no forman parte del tamaño de una trama. En 1998, el estándar IEEE 802.3a amplía el tamaño máximo de la trama a 1522 bytes para adaptarse a un tipo de tecnología de redes conmutadas conocidas como: red de área local virtual VLAN (Virtual Local Area Network).

Las diferencias entre los estilos de tramas IEEE 802.3 (original y el revisado) consisten en agregar

un delimitador de inicio en la trama SFD (*Start Frame Delimiter*) y el cambio en el Tipo de Campo que incluye longitud.

Preámbulo

El **preámbulo** es una serie de 8 octetos que preceden al paquete de datos de la capa física; tiene la finalidad de permitir que las estaciones receptoras sincronicen sus relojes con el mensaje entrante a fin de leerlo sin errores. El último byte se denomina delimitador de comienzo de marco SFD.

Direcciones origen y destino

Las direcciones **destino** y **origen** son direcciones físicas, es decir, corresponden a dispositivos reales o adaptadores de red NIC. Por lo tanto, la dirección destino se refiere al NIC que recibirá el datagrama (*recipient address*) y la dirección origen se denomina *source address*.

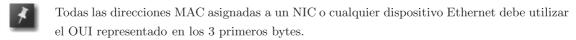
Dirección MAC

La dirección MAC (*Media Access Control*) de un dispositivo físico (NIC) es la dirección real de cualquier computadora en la red. Por ejemplo, a pesar de que el protocolo TCP/IP (*Transmission Control Protocol/Internet Protocol*) utiliza un sistema de direcciones lógicas (denominadas direcciones IP), estas direcciones deben ser traducidas a las direcciones MAC de los adaptadores de red a donde van dirigidos los mensajes.

Cada NIC tiene un número de identificación (dirección) de 6 bytes que es único en el mundo y no se repite (representa huella dactilar del dispositivo), esta dirección está contenida en el hardware de la tarjeta o adaptador de red y no debe ser alterado. Los 6 bytes de los campos de dirección suelen indicarse en formato hexadecimal, como por ejemplo: 00-10-C3-11-FC-CA.

Los fabricantes de este tipo de tarjetas tienen que solicitar a la IEEE asignación de un número de 24 bits (3 bytes), que les es remitido, y que sirve para identificar las tarjetas del fabricante a partir de ese momento. Es el OUI (*Organizationally Unique Identifier*) conocido como código de vendedor. A continuación cada fabricante añade a su OUI otros 24 bits, hasta totalizar 48 (6 octetos), en los que se puede incluir cualquier información que se desee, desde datos de fabricación hasta características de la tarjeta.

Con la finalidad de garantizar direcciones únicas a cada dispositivo Ethernet, el valor de la dirección MAC es el resultado directo de las normas implementadas para proveedores que se encuentran registrados ante el IEEE, respetando lo siguiente:





IEEE asigna a cada proveedor un código de 3 bytes llamado **identificador único organizacional** OUI la dirección MAC es parte de una PDU en la capa 2, consta de un valor binario de 48 bits expresado como 12 dígitos hexadecimales

La dirección MAC es un dato grabado en forma permanente en la memoria ROM del dispositivo NIC; cuando se inicializa este dispositivo se transfiere una copia de la dirección MAC a la memoria RAM para ser utilizada como dirección de origen para compararla con la dirección destino y de esta forma determinar si un mensaje debe pasarse a las capas superiores para su procesamiento.

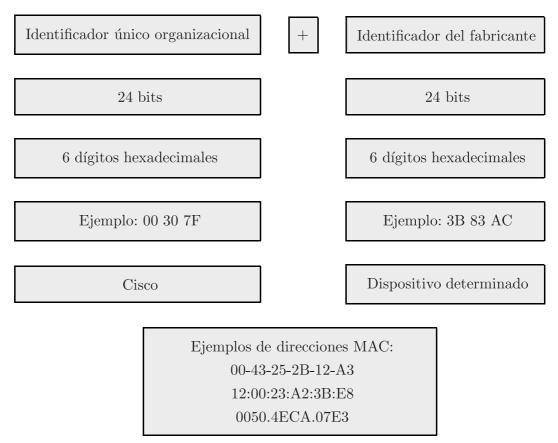


Figura 11.3 Distintas representaciones para direcciones MAC.

Dependiendo del fabricante el formato de direcciones MAC no es único, por ejemplo: 00-32-1C-8A-

77-56, 11:03:91:A5:23:00 o 0012.23AD.6500

Las direcciones MAC se asignan a cualquier dispositivo que pueda transmitir y recibir datos en la red como son:

Estaciones de trabajo work station.

Servidores y computadoras.

Impresoras.

Switchs y routers.

Módulos remotos, tablets, consolas de video juegos, teléfonos móviles o celulares, etc.

Código de tipo

Este campo está formado por 16 bits que se utilizan para identificar el tipo de protocolo de alto nivel que está siendo empleado en la red Ethernet. Señala por tanto el tipo de dato que está siendo transportado en el campo de datos del paquete o frame.

Campo de datos

El campo de datos del datagrama puede variar entre un mínimo de 46 y un máximo de 1500 bytes, así que la longitud máxima de un paquete Ethernet es de 1518 bytes, y 64 la mínima. Cuando una estación transmite un datagrama mayor que los 1518 bytes permitidos (equivale a una transmisión de más de 20 milisegundos), ocurre una condición de error denominada Jabber, el datagrama resultante se denomina "Long Frame". Si el paquete tiene una longitud menor que la mínima, también es una condición errónea, a pesar que FCS sea correcto y se denomina "Short Frame".

Verificación de integridad

El campo FCS contiene 32 bits "checksum" del frame o paquete de datos, se utiliza para detectar errores en la trama, emplea comprobación cíclica redundante CRC (Cyclical Redundancy), el dispositivo emisor incluye los resultados del CRC en el campo FCS de la trama.

Cuando el dispositivo receptor recibe la trama, calcula su propio CRC para detectar errores, si los CRCs coinciden significa libre de errores, en caso contrario se descarta la trama. La figura 11.4 muestra la secciones de campos incluidos en la verificación de la trama de Ethernet.

| IEEE 802.3 | | ← Cam | | | | |
|------------|--------------------------------------|----------------------|---------------------|-----------------|-----------------------|--------------------------|
| 7 | 1 | 6 | 6 | 2 | 46 a 1500 | 4 |
| Preámbulo | Delimitador de inicio de trama | Dirección destino | Dirección origen | Longitud y tipo | Encabezado y datos | Verificación de trama |

Figura 11.4 Estructura y tamaño en bytes de cada campo de una trama.

11.4 Arduino Ethernet Shield



Da Entro de las características de los modelos de tarjetas Arduino, se encuentra la conexión a Ethernet, esto se logra usando la interface electrónica Arduino Ethernet Shield, la cual se basa en el chip W5100 de la compañía WIZnet; este chip permite implementar los protocolos Ethernet y TCP asegurando que los datos se transmitan o reciban correctamente. La conexión se realiza mediante un cable estándar CAT5 o CAT6 con conector tipo jack RJ45 y la velocidad de conexión se encuentre en 10/100 Mb. La tarjeta Arduino emplea de manera interna el puerto SPI para enlazarse con el chip W1500.

La figura 11.5 muestra el diagrama a bloques del dispositivo W5100. Como parte de las características técnicas de Arduino Ethernet Shield se encuentran las siguientes: compatibilidad con el estándar IEEE802.3af, bajo nivel de rizo y ruido de la señal de salida (100 mVpp), el rango de voltaje de entrada es de 36 V a 57 V, protección para cortocircuito y sobrecargas y 9 V de salida; el botón de reset sobre la tarjeta Shield reinicializa al chip W5100 como a la tarjeta Arduino. Sobre la tarjeta Arduino Ethernet Shield se encuentran una serie de diodos LEDs que presentan información como: PWR indica que la tarjetas Arduino y Shield se encuentran

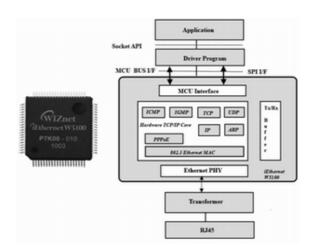


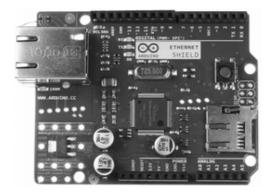
Figura 11.5 Diagrama a bloques del circuito integrado W5100.

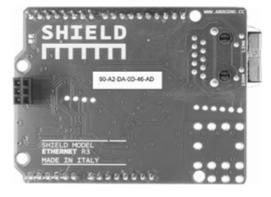
con fuente de alimentación, LINK describe la presencia de transmisión/recepción de datos por internet, FULLD indica que la conexión de la red es del tipo full-duplex, 100M exhibe la presencia

de la velocidad en la red a 100 Mb, Rx y Tx exhiben luz en forma intermitente cuando reciben y envían datos, respectivamente. COLL indica si existen colisiones en la red.

Sobre la tarjeta Ethernet Shield viene un puente (jumper) marcado con la etiqueta INT (conecta el pin INT de W5100 al pin digital 2 de la tarjeta Arduino) el cual permite que la tarjeta Arduino reciba notificaciones de eventos de internet por interrupción (esta opción no está soportada por la librería de Ethernet). La comunicación de la tarjetas Arduino con W5100 se realiza utilizando los pins digitales (10, 11, 12 y 13 para el modelo UNO y los pins 50, 51 y 52 para Mega); en ambos modelos UNO y Mega el pin 10 se utiliza para habilitar el dispositivo W5100, además las señales del puerto SPI están empleadas para comunicación interna y por lo tanto no se pueden utilizar.

En la figura 11.6 se muestra la cara superior e inferior del circuito impreso de la tarjeta Arduino Ethernet Shield; note la dirección MAC que se presenta en la figura 11.6b, la cual se encuentra guardada en la memoria ROM como parte de la información importante de esta interface.





(a) Ethernet Shield cara superior.

(b) Ethernet Shield cara inferior.

Figura 11.6 Arduino Ethernet Shield.

Conectando a internet el Sistema Arduino

Para acceder a los servicios de internet con la tarjeta Ethernet Shield y los modelos de tarjeta Arduino compatibles con esta interface (por ejemplo, los modelos UNO, Due, Mega, Leonardo, YUN) es necesario realizar el siguiente procedimiento:



Conectar la tarjeta Arduino Ethernet Shield sobre la tarjeta Arduino.



La tarjeta Arduino se conecta a la computadora por medio del cable USB.



La tarjeta Ethernet Shield se conecta a un switch o router utilizando un cable Ethernet

Alfaomega

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

Fernando Reyes Cortés • Jaime Cid Monjaraz

estándar CAT5 o CAT6 con conector jack RJ45.

- La dirección MAC de la tarjeta Ethernet Shield es un identificador único que regularmente se encuentra en la parte posterior del circuito impreso como se indica en la figura 11.6b
- La dirección IP se puede obtener en la consola de comandos del sistema operativo Windows, a través del comando **ipconfig**. El procedimiento para encontrar esta información se describe en el cuadro 11.1 que se documenta en el ejemplo 11.1.
- Una vez determinadas las direcciones MAC e IP, entonces se programa la tarjeta Ethernet Shield mediante la función **Ethernet.begin(mac,ip)** (ver sección 11.5 Librería Ethernet).

A través de Ethernet, el sistema Arduino expande sus características potenciales de control, automatización, procesamiento de señales y en general múltiples aplicaciones en ingeniería y ciencias exactas. La figura 11.7 muestra el concepto global para cubrir aplicaciones remotas desde cualquier parte del mundo (con servicio de red); la conexión se puede realizar a un switch, router, servidor, computadora o a cualquier dispositivo conectado a Ethernet para realizar intercambio de información.



Figura 11.7 Conectividad del Sistema Arduino.

Este aspecto representa una enorme ventaja que tienen las tarjetas Arduino, ya que además de ser sistemas empotrados, es decir, que tienen el objetivo específico de ejecutar un sketch sobre la tarjeta Arduino para automatizar un proceso, sin perder este rasgo distintivo, también cuentan con la características de conectividad y enlace a la tecnología más utilizada a nivel mundial.



11.5 Librería Ethernet

La tarjeta Ethernet Shield requiere de la librería Ethernet para configurar el protocolo de comunicación de información en la red, es decir tener una correcta recepción y transmisión de la información. Adicionalmente, también se utiliza la librería SPI (para comunicación con dispositivos usando interface periférica serial), ambas librerías se declaran en las primeras líneas del skecth (es decir, en el header o cabecera). El módulo Wiznet de la tarjeta Ethernet Shield se comunica en forma interna con la tarjeta Arduino a través del protocolo SPI.

A continuación se describen la sintaxis de las funciones que forman parte de la librería Ethernet.



Ethernet.begin(mac);

Inicializa la librería Ethernet y configura la red. Con esta función, la tarjeta Ethernet Shield automáticamente obtiene la dirección IP. El argumento de entrada **mac** (Media Access Control) representa la dirección para el dispositivo, es decir la dirección de hardware de la tarjeta Shield y consta de un arreglo de 6 datos del tipo byte.

En la cara posterior de la tarjeta Ethernet Shield se encuentra localizada la dirección MAC sobre una etiqueta como se ilustra en la figura 11.6b.

La función **Ethernet.begin(...)** también acepta las siguientes formas de sintaxis:



Ethernet.begin(mac, IP);

Donde el argumento de entrada **IP** representa la dirección del dispositivo y está formado de un arreglo de 4 bytes.

La versión 1.0 de esta función retorna el valor $\mathbf 1$ o verdadero cuando la conexión se realiza correctamente; retorna un valor $\mathbf 0$ o falso si se presenta alguna falla. Para otro tipo de versiones no retornan información.



Ethernet.localIP();

Obtiene la dirección de la tarjeta Ethernet Shield. No requiere de parámetros o argumentos de entrada, retorna la dirección IP.

Ethernet.maintain();



Esta función no tiene argumento de entrada, permite renovar la conexión DHCP, cuando se asignan direcciones IP a través de DHCP, dispositivos Ethernet tienen la dirección por un determinado tiempo, por lo tanto con **Ethernet.maintain()** se puede solicitar la renovación del servidor DHCP. Dependiendo de la configuración del servidor, la función puede retornar la misma dirección, nueva o ninguna. Retorna: 0 cuando no ha sucedido nada, 1 renovación no otorgada, 2 renovación exitosa, 3 vinculación rechazada y 4 para vinculación otorgada.

IPAddress(address);



Define una dirección IP, puede ser usado para declarar direcciones locales o remotas; el argumento de entrada **address** está compuesto por 4 bytes separados por comas, por ejemplo: **IPAddress**(192, 165, 1,1). Esta función no retorna datos.

Œ

11.5.1 Ethernet: EthernetServer

Genera servidores definidos por el usuario pertenecientes a **Server class**, es decir, es un tipo particular de clases para servidores con la estructura adecuada para enviar/recibir datos a clientes conectados sobre otros dispositivos o computadoras.

EthernetServer();



Crea un objeto tipo servidor que "escuche" las conexiones entrantes de un puerto específico. La sintaxis que se utiliza es:

Server(port);

donde el argumento de entrada **port** es un dato tipo entero (int) y representa el puerto a "escuchar". No retorna información.

server.available();



Obtiene un cliente que esté conectado al servidor y que tenga datos disponibles para lectura; la

conexión persiste cuando el cliente retornado por la función se sale de ámbito. La conexión se puede cerrar empleando la función client.stop().

La función server.available() no tiene parámetros o argumentos de entrada y retorna un cliente objeto. Si ningún cliente tiene datos disponibles para lectura, este objeto se evalúa como una condición falsa dentro de una instrucción $if(...)\{...\}$ else $\{...\}$; available() se hereda de la clase Stream.



Escribe (transmite) datos a todos los clientes conectados al servidor. El argumento de entrada val indica el valor enviado como un simple byte.

Los datos pueden ser enviados como un simple byte o una serie de bytes, por lo que, la función server.write(val) también admite la siguiente sintaxis:

```
\infty
                                       server.write(buf, len);
```

Donde el argumento buf representa un arreglo de datos a transmitir como una serie de bytes y len indica el tamaño o longitud del arreglo. La función server.write() retorna el número de bytes enviados.

```
∞
                                 server.printf(data);
```

Imprime un dato como una secuencia de dígitos a todos los clientes conectados al servidor, por ejemplo: el número 893 es enviado como tres caracteres consecutivos '8', '9' y '3'.

El argumento data puede ser del tipo char, byte, int, long o string (cadena de caracteres). También de manera opcional se puede especificar la base del número definido en data, en tal caso admite la siguiente sintaxis:

```
∞
                               server.printf(data, base);
```

Donde base indica la base del argumento data: BIN (binario, base 2), DEC (decimal, base 10),

OCT (octal, base 8), HEX (base hexadecimal, base 16). La función **server.printf(...)** retorna el número de bytes escritos.

server.printfln(data);



Imprime data seguido de una nueva línea a todos los clientes conectados al servidor; esta función trabaja de una forma similar a server.printf(...). Admite las siguientes variantes de sintaxis:



server.printfln(); //transmite una línea en blanco.



server.printfln(data); //data: char, byte, int, long o string (cadena de caracteres).



server.printfln(data, base); //base indica el tipo de número de data.



11.5.2 Ethernet: Client class

Genera clientes o navegadores que pueden conectarse a servidores para enviar y recibir datos, representa la base para todas las llamadas de clases de funciones definidas por el usuario.

Por ejemplo: **EthernetClient()** crea un cliente para conectarse a una dirección IP y puerto específico de internet.

Para este caso la sintaxis con la que se utiliza es:

EthernetClient();

Esta función no cuenta con argumentos o parámetros de entrada.

Para generar un objeto cliente se procede de la siguiente manera:

EthernetClient client;

$if(client)\{...\}$

indica si el cliente especificado por **EthernetClient** está listo. No tiene parámetros de entrada y retorna un valor booleano verdadero cuando el cliente especificado está disponible.

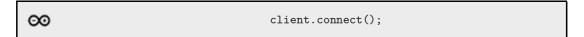
Nota: el usuario puede declarar el nombre del identificador para la clase de cliente que considere más adecuado; sin embargo, de acuerdo a la definición anterior, se utilizará client para describir la sintaxis de las siguientes funciones:

client.connected();



Determina si el cliente **client** se encuentra conectado. Se considera que un cliente está conectado aun sin la conexión ha sido cerrada, pero el dato no se ha leído.

Esta función no tiene argumentos de entrada y retorna un valor booleano verdadero si el cliente está conectado, falso si no lo está.



Conecta el cliente a una dirección IP y puerto especificado, retornado un valor booleano verdadero cuando se establece la conexión (falso en caso de rechazo). También soporta DNS cuando usan un nombre de dominio.

Esta función acepta las siguientes formas de sintaxis:

- client.connect();
- client.connect(ip, port);
- client.connect(URL, port);

donde **ip** indica la dirección IP que el cliente se conectará (consta de un arreglo de 4 bytes), **URL** el nombre del dominio que el cliente se conectará, por ejemplo, puede ser una cadena de caracteres de la forma "arduino.cc" y **port** es un dato de tipo entero (int) indicando el puerto que el cliente se conectará.

```
Client.write(val);
```

Escribe (transmite) datos al servidor cuando el cliente está conectado. El argumento de entrada val puede ser enviado como un simple byte o una serie de bytes. También acepta la siguiente sintaxis:

```
client.write(buf, len);
```

El argumento de entrada **buf** representa un arreglo de bytes y **len** es la longitud del arreglo. Retorna el número de bytes transmitidos.

ALFAOMEGA ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA FERNANDO REYES CORTÉS • JAIME CID MONJARAZ

∞

client.printf(data), client.printf(data, base)

Estas funciones trabajan de manera similar a las ya descritas en la sección referente al servidor: server.printf(data) y server.printf(data, base) (ver la página 396).

client.available();



Retorna la cantidad de datos disponibles que han sido escritos al cliente (debe estar conectado) por el servidor; esta función no tiene argumentos de entrada, available() hereda la forma de clase Stream.

client.read();



Lee el siguiente byte recibido del servidor (el cliente debe estar conectado). El dato estará disponible después de la última llamada de lectura).

Esta función hereda las características de **Stream class**. y no tiene parámetros o argumentos de entrada, retorna el siguiente byte o carácter, cuando devuelve un valor -1 significa que los datos no están disponibles.

client.flush();



Descarta cualquier byte que ha sido escrito al cliente y que aún no han sido leídos. Esta función no contiene parámetros de entrada y tampoco retorna datos; flush() hereda todas las características de Stream class.

client.stop();



Esta función desconecta al cliente **client** del servidor, no tiene parámetros de entrada y ningún tipo de retorno de datos.



Configuración Cliente

La tarjeta Ethernet Shield puede ser configurada como cliente para solicitar recursos a un servidor. En el sitio Web de este libro se encuentra la programación básica para configurar al sistema Arduino como cliente y su enlace a un servidor para desplegar información de sensores, variables de control y automatización de procesos en una página Web.



Configuración Servidor

Otra de las ventajas del sistema Arduino, es que puede ser configurado como servidor, de esta forma acepta solicitudes de navegadores para enviar y recibir datos de instrumentación, control y procesamiento. En el sitio Web de este libro se encuentran ejemplos prácticos que muestran aplicaciones del sistema Arduino configurado como servidor.



11.6 Ejemplos prácticos

La saplicaciones de las tarjetas Arduino no sólo incluyen procesos en forma local enfocados a instrumentación, procesamiento de datos, control automático, robótica y mecatrónica, también abarcan aplicaciones de manera global a través de la tecnología de Ethernet; en esta sección se presenta un conjunto de ejemplos prácticos donde se muestra el potencial del Sistema Arduino para enviar/recibir información desde cualquier parte del mundo (con servicio de red) usando la tarjeta Ethernet Shield con soporte de las funciones de la librería Ethernet.

A continuación se describen 3 casos de estudios para mostrar la facilidad que tiene el sistema Arduino para conectarse a direcciones IP para desplegar o manipulador datos. El primer ejemplo muestra la programación básica para desplegar el letrero "Hola mundo" en una página Web; los dos ejemplos restantes exhiben el resultado de sumar dos matrices de dimensión de 3×3 y adquisición de datos de un sensor de voltaje.

Para llevar a cabo dicho ejemplos se requieren los siguientes componentes (ver figura 11.8):



Tarjeta Ethernet Shield.

Alfaomega

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

Fernando Reyes Cortés • Jaime Cid Monjaraz

- Tarjeta Arduino (pueden ser los modelos UNO, Mega, Leonardo y Due) con cable USB .
- Cable RJ45 con conector macho.
- Switch o router.
 - Computadora con conexión a internet (inalámbrica o mediante cable RJ45) y ambiente de programación Arduino correctamente instalado, para compilar y ejecutar los sketchs en la tarjeta Arduino.



Figura 11.8 Componentes requeridos para realizar los ejemplos prácticos.

♣ Ejemplo 11.1

Utilizar la tarjeta Ethernet Shield y el modelo Arduino UNO para desplegar en una página de internet el mensaje "Hola mundo".

Solución

Por medio de un sencillo procedimiento se pretende mostrar el potencial que tiene el Sistema Arduino en aplicaciones de Ethernet. Un ejemplo básico de programación es el desplegado del mensaje "Hola mundo" en una página Web asignada a la tarjeta Ethernet Shield manejada por los modelos Arduino que admiten esta interface. Con tal finalidad, es necesario que la computadora que se utiliza para programar la tarjeta Arduino esté conectada a internet, debido a que se requiere conocer la dirección IP que proporciona la red. Para el sistema operativo Windows, directamente en la consola de comandos teclear **ipconfig** \hookleftarrow como se ilustra en el cuadro 11.1 (para encontrar la consola de comandos, utilice el menú de programas de Windows, sobre el diálogo o recuadro **buscar**, teclee **cmd**).

Para finalidades ilustrativas se empleará la dirección indicada en IPv4 192.168.1.102, esta información dependerá del servicio y características de la red de área local. Con este dato, a la tarjeta Ethernet Shield se le asignará una dirección IP que esté libre en la misma red, por ejemplo se puede modificar el último dígito, obteniendo: 192.168.1.101; el usuario debe comprobar que esta dirección esté libre ejecutando el programa **cap11_WebBasico** (ver cuadro de código Arduino 11.1), de no ser así, puede probar con las siguientes opciones: 192.168.1.103 o 192.168.1.104; también se requiere la dirección MAC de la tarjeta Ethernet Shield, la cual se encuentra disponible sobre una etiqueta colocada en la parte posterior del circuito impreso como se muestra en la figura 11.6b.

c:\Users\ipconfig

Configuración IP de Windows

Adaptador de Ethernet:

Sufijo DNS específico por conexión:

 Vínculo: dirección IPv6 local: ...
 fe80:55c9:9c24:be48:839e%12

 Dirección IPv4:
 192.168.1.102

 Máscara de subred:
 255.255.255.0

 Puerta de enlace predeterminada
 192.168.1.1

Cuadro 11.1 Comando ipconfig en la consola de Windows 8.1.

El código Arduino 11.1 presenta la programación del sketch **cap11_WebBasico** para realizar la comunicación Ethernet usando la tarjeta Shield con los modelos de tarjeta Arduino que aceptan esa interface como son: UNO, Mega, Due, Yun y Leonardo. En el header del programa, en las dos primeras líneas se encuentran las librerías **SPI** y **Ethernet** para llevar a cabo el protocolo de comunicación; las funciones de la librería Ethernet utilizan de manera interna el protocolo serial peripheral interface, por lo que se requiere la librería SPI.

La dirección MAC de la tarjeta Ethernet Shield se encuentra en el respaldo del circuito impreso (ver figura 11.6b), esta información se utiliza para inicializar un arreglo del tipo byte con 6 elementos, como se presenta en línea 3. La dirección IP depende de las características de configuración de la red de área local y se obtiene como se ilustra en el cuadro 11.1 a través del comando **ipconfig** de la consola de Windows (línea 5.)

La declaración del servidor se realiza en la línea 6, se utiliza la clase **EthernetServer** por medio del puerto 80 para enviar y recibir datos a clientes conectados a la página Web 192.168.1.101; a partir de la línea 8 se encuentra la subrutina de configuración **setup()**, donde se inicializa la velocidad de comunicación serial en 9600 Baudios entre la tarjeta Arduino y la computadora que contiene el ambiente integrado de programación Arduino; también se inicia la conexión Ethernet y

se proporciona en el monitor serial Arduino la dirección IP asignada a la tarjeta Ethernet Shield (línea 13). La subrutina principal de programación loop() inicia a partir de la línea 15.



Figura 11.9 Entorno integrado Arduino.

La línea 16 contiene la declaración del cliente para conectarse al servidor con la estructura requerida para enviar y recibir datos; **EthernetClient()** crea un cliente para conectarse a la dirección IP 192.168.1.101 usando el puerto 80. Observe que en la línea 18 indica si el cliente se encuentra listo, si el valor booleano es verdadero, entonces la información enviada por el servidor se lee en la línea 19 para que se transmita al monitor serial Arduino (ver figura 11.9). En la línea 22 se envía a la página Web 192.168.1.101 el mensaje de bienvenida "Hola mundo",

tal y como se muestra en la figura 11.10, después de 4 segundos se desconecta el sistema Arduino a dicha página Web, el proceso se repite de manera indefinida por la subrutina **loop()**.

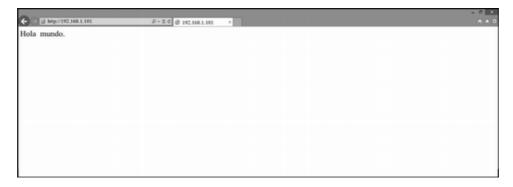


Figura 11.10 "Hola mundo" en la página Web asignada a Ethernet Shield.

```
∞ Código Arduino 11.1: sketch cap11_WebBasico
  Arduino. Aplicaciones en Robótica y Mecatrónica.
  Capítulo 11 Ethernet.
  Fernando Reyes Cortés y Jaime Cid Monjaraz.
  Alfaomega Grupo Editor: "Te acerca al conocimiento".
   Sketch cap11_WebBasico.ino
 1 #include <SPI.h>//protocolo: serial peripheral interface.
 2 #include <Ethernet.h>//funciones para enlace de comunicación Ethernet.
 3 byte mac[] = \{0x90, 0xA2, 0xDA, 0x0D, 0x46, 0xAD\};//MAC ver figura 11.6b.
4 //La direcciín IP se obtiene utilizando el comando ipconfig (ver cuadro 11.1).
 5 IPAddress ip(192,168,1,101); //dirección IP 192.168.1.101
 6 EthernetServer server(80);//puerto por default 80.
 7 //Subrutina de configuración.
8 void setup() {//configura comunicación.
       Serial.begin(9600);
9
       Ethernet.begin(mac, ip);//inicia conexión Ethernet.
10
11
       server.begin();
12
       Serial.print("Ip del Servidor");
       Serial.println(Ethernet.localIP());
13
14 }
15 void loop() {//lazo principal de programación.
       EthernetClient client = server.available();//escuchar clientes.
16
       //"Escucha" clientes conectados al servidor.
17
18
       if (client) {//cliente conectado.
          char c = \text{client.read}();//\text{lee} información del servidor.
19
          Serial.write(c);// transmite información al monitor serial.
20
          //Se transmite a la página Web el mensaje de bienvenida: "Hola mundo".
21
          client.println("Hola mundo. <br/> />" );//navegador Web recibe datos.
22
          client.println("</html>");//código htlm.
23
24
          delay(4000);//pausa en espera antes de cerrar conexión.
          client.stop();//cierra conexión de Ethernet.
25
          Serial.println("Cliente desconectado");
26
27
28 }
```

Una ventaja importante de Ethernet es que desde cualquier dispositivo conectado a la página Web asignada a la tarjeta Ethernet Shield puede visualizar la información; los teléfonos móviles inteligentes (celulares), tablets, Ipads, computadoras, consolas de videos juegos, etc., son algunos ejemplos de dispositivos que desde cualquier parte del mundo (con servicio internet) pueden acceder a la información, procesamiento y manipulación de datos realizada por las tarjetas Arduino.



HTTP y HTML

El protocolo HTTP (HyperText Transfer Protocol) es un formato básico y universal que utilizan los clientes o navegadores Web para comunicarse con servidores de la red. Este protocolo consiste en solicitudes concretas que realiza un cliente Web y respuestas predefinidas que ofrece el servidor Web. HTTP define la sintaxis y contexto de los elementos de software de la arquitectura Web para comunicarse entre servidores y clientes.

HTLM (HyperText Markup Language) es un lenguaje de programación de las páginas Web; cuando un cliente realiza una solicitud a un servidor Web se realiza por medio de código fuente HTLM, indicando la definición, características y contenido de las páginas Web, tales como: dimensiones de los márgenes, tipos de letras, fotos y videos, etc.



Ejemplos Ethernet

En el sitio Web de este libro se encuentran ejemplos adicionales que ilustran el potencial del Sistema Arduino para comunicarse por Ethernet a plataformas de redes sociales tales como: Google+, Twitter, facebook, etc.

Una herramienta útil del sistema operativo Windows que permite verificar la configuración de la tarjeta Ethernet Shield en la red es por medio del comando **Ping**. En la consola de comandos teclear lo siguiente: **Ping -a 192.168.1.101** (recuerde que la dirección IP depende de las características de la red local); el cuadro 11.2 muestra la información que despliega el comando **Ping**, el cual envía paquetes de datos a la dirección IP, también se muestra las estadísticas de paquetes enviados, recibidos e información perdida, así como el tiempo de comunicación de ida y vuelta. Este comando proporciona certeza de que la dirección IP asignada a la tarjeta Ethernet Shield se encuentra libre en la misma red. Observe que los diodos LED Rx y Tx sobre la tarjeta Shield se encienden con el envío/recepción de datos.

c:\Users\Ping -a 192.168.1.101

Haciendo ping a 192.168.1.101 con 32 bytes de datos

Respuesta desde 192.168.1.101: bytes=32 tiempo=1 ms TTL=128 Respuesta desde 192.168.1.101: bytes=32 tiempo<1 ms TTL=128 Respuesta desde 192.168.1.101: bytes=32 tiempo<1 ms TTL=128 Respuesta desde 192.168.1.101: bytes=32 tiempo=1 ms TTL=128

Estadísticas de ping para 192.168.1.101: Paquetes: enviados = 4, recibidos = 4, perdidos = 0 < 0% perdidos>.

Tiempos aproximados de ida y vuelta en milisegundos Mínimo = 0ms, Máxima=1ms, Media=0ms.

Cuadro 11.2 Comando Ping de Windows 8.1.

Procedimiento para ejecutar el sketch cap11_WebBasico

Una vez ensamblada la tarjeta Shield en algún modelo Arduino, computadora, cable RJ45, cable USB, conexión a red, el procedimiento para ejecutar correctamente el sketch **cap11_WebBasico** se describe a continuación:

- Editar el sketch **cap11_WebBasico** del cuadro de código Arduino 11.1.
- Compilar el sketch **cap11_WebBasico** mediante el icono .
- Descargar el código de máquina a la tarjeta Arduino usando 👈.
- Abrir la ventana del monitor serial del ambiente Arduino usando el icono 🔑
- Abrir un navegador de páginas Web e ingresar el IP asignado a la tarjeta Arduino Ethernet Shield.

♣ ♣ Ejemplo 11.2

Implementar en el sistema Arduino el algoritmo para sumar matrices y desplegar el resultado en una página Web. Considere las siguientes matrices:

$$A = \begin{bmatrix} 1 & 2 & 2 \\ 4 & 8 & 9 \\ 6 & 3 & 1 \end{bmatrix} \qquad B = \begin{bmatrix} 7 & 6 & 9 \\ 4 & 6 & 1 \\ 2 & 5 & 2 \end{bmatrix}$$

Obtener C = A + B.

Solución

El cuadro de código Arduino 11.2 describe el sketch **cap11_WebServer** con la programación necesaria para desplegar en una página Web asignada a la tarjeta Ethernet Shield el resultado C = A + B, donde $A, B, C \in \mathbb{R}^{3 \times 3}$.

Las líneas 1 y 2 contienen las librerías de protocolo SPI (serial peripheral interface) y Ethernet, respectivamente; a partir de la línea 4 a la 11 se encuentra la declaración de las matrices A, B, C.

La dirección MAC de la tarjeta Ethernet Shield se declara en la línea 14 (ver figura 11.6b) y la dirección IP se obtiene de acuerdo al procedimiento indicado en el cuadro 11.1; el puerto de enlace con la página Web es 80.

La subrutina de configuración **setup()** se encuentra en la línea 19, establece la velocidad de comunicación en 9600 Baudios con la computadora, así como la configuración de Ethernet. La subrutina principal de programación **loop()** inicia en la línea 26.

Clientes disponibles en la página Web se detectan en la línea 27, envían solicitudes de configuración por medio de comandos HTLM (líneas 35-42).

El algoritmo para sumar matrices se lleva a cabo en la línea 44 y el envío de información para desplegar el resultado en la página Web se realiza a partir de la línea 48. Posteriormente se hace una pausa de un milisegundo antes de desconectar al cliente. Este proceso se realiza de manera indefinida por la subrutina principal **loop()**.

A través de un navegador de páginas Web, puede acceder a la dirección IP de la tarjeta Ethernet Shield se puede visualizar el resultado del procesamiento de la tarjeta Arduino como se muestra en la figura 11.11; observe que la información que se recibe del servidor se despliega en el monitor serial del ambiente de programación Arduino.

```
∞ Código Arduino 11.2: sketch cap11_WebServer
   Arduino. Aplicaciones en Robótica y Mecatrónica.
   Capítulo 11 Ethernet.
   Fernando Reyes Cortés y Jaime Cid Monjaraz.
   Alfaomega Grupo Editor: "Te acerca al conocimiento".
   Sketch cap11_WebServer.ino
 1 #include <SPI.h> //protocolo: serial peripheral interface.
 2 #include <Ethernet.h>//librería Ethernet para enlace de comunicación.
 _{\mathbf{3}} //Matriz A = \begin{bmatrix} 1 & 2 & 2 \\ 4 & 8 & 9 \\ 6 & 3 & 1 \end{bmatrix}
 4 float A[3][3] = \{1,2,2,
                    4,8,9,
                    6,3,1};
   //Matriz B = \begin{bmatrix} 7 & 6 & 9 \\ 4 & 6 & 1 \\ 2 & 5 & 2 \end{bmatrix}
 8 float B[3][3] = \{7,6,9,
 9
                   4,6,1,
10
                   2,5,2};
11 float C[3][3];//la variable C almacena el resultado A+B.
12 //La dirección MAC de la tarjeta Ethernet Shield se encuentra en el respaldo del circuito impreso.
13 // En el caso de la tarjeta Ethernet Shield utilizada es: 90-A2-DA-0D-46-AD
14 byte mac[] = {0x90, 0xA2, 0xDA, 0x0D, 0x46, 0xAD };// asigna dirección MAC.
15 //La dirección IP se obtiene por medio del comando ipconfig, ver cuadro 11.1.
16 //Para este ejemplo la dirección IP utilizada es: 192.168.1.101
17 IPAddress ip(192,168,1,101); //Dirección IP
18 EthernetServer server(80);//Por default el puerto HTTP es 80.
19 void setup() {//subrutina de configuración.
20
        Serial.begin(9600);
        Ethernet.begin(mac, ip);//Inicia conexión Ethernet con el servidor.
21
22
        server.begin();
        Serial.print("Ip del Servidor");
23
        Serial.println(Ethernet.localIP());
24
25 }
26 void loop() {
        EthernetClient client = server.available();//clientes entrantes.
```

Continúa código Arduino 11.2a: sketch cap11_WebServer

Arduino. Aplicaciones en Robótica y Mecatrónica.

Capítulo 11 Ethernet.

Fernando Reyes Cortés y Jaime Cid Monjaraz.

Alfaomega Grupo Editor: "Te acerca al conocimiento".

Continuación del sketch cap11_WebServer.ino

```
if (client) {
28
           Serial.println("Nuevo cliente");
29
           boolean currentLineIsBlank = true;
30
           while (client.connected()) {
31
32
              if (client.available()) {
                 char c = client.read();
33
                 Serial.write(c);
34
                 if (c == '\n' && currentLineIsBlank) {
35
                     client.println("HTTP/1.1 200 OK"); //encabezado respuesta HTTP.
36
                     client.println("Content-Type: text/html" );
37
38
                     client.println("Connection: close");//cerrar conexión después de respuesta.
                     client.println("Refresh: 5"); //actualiza página cada 5 segundos.
39
                     client.println();
40
                     client.println("<!DOCTYPE HTML>" );
41
                     client.println("<html>" );
42
                     for(int i=0; i<3; i++){//Algoritmo para sumar matrices C=A+B.
43
                         for(int j=0; j<3; j++){ C[i][j]=A[i][j]+B[i][j];}
44
45
                     client.println("Ejemplo cap11_WebServer: Algoritmo para sumar matrices. <br/> <br/> />" );
46
                     client.println("<PRE>
                                                   С
                                                               Α
                                                                             B <PRE />" );
47
                     for (int i=0; i<3; i++) {//desplegado de las matrices en página Web.
48
                             for(int j=0; j<3; j++){//envía matriz C \in \mathbb{R}^{3\times 3}.
49
                                client.print(C[i][j]); client.print(" ");
50
51
                             for(int j=0; j<3; j++){//envía matriz A \in \mathbb{R}^{3\times 3}.
52
                                client.print(A[i][j]); client.print(" ");
53
54
```

```
Continúa código Arduino 11.2b: sketch cap11_WebServer
  Arduino. Aplicaciones en Robótica y Mecatrónica.
  Capítulo 11 Ethernet.
  Fernando Reyes Cortés y Jaime Cid Monjaraz.
  Alfaomega Grupo Editor: "Te acerca al conocimiento".
   Continuación del sketch cap11_WebServer.ino
                        for(int j=0; j<3; j++){//envía matriz B \in \mathbb{R}^{3\times 3}.
55
                           client.print(B[i][j]); client.print(" " );
56
57
                        client.println("<br/>");
58
                  }// fin del for(;;){...} línea 48.
59
                  client.println("</html>" );
60
                  break;// condición de salida de while(client.connected()){...} línea 31.
61
               62
               if (c == ' \setminus n') 
63
64
                  currentLineIsBlank = true;//comienza nueva linea.
65
               else if (c != '\r') {
66
                  currentLineIsBlank = false;//fin de línea, cuando hay retorno de carro.
67
68
            69
         }//fin de while(client.connected()){...} de la línea 31
70
         delay(1);//pausa de un milisegundo.
71
```

El lector también puede comprobar el resultado conectándose a la dirección IP de la tarjeta Ethernet Shield por medio de una computadora con conexión a red, teléfono celular del tipo smartphone, tablet o Ipad. El procedimiento para ejecutar el sketch **cap11_WebServer** y desplegar el resultado de operaciones aritméticas con matrices en una página Web es similar al descrito en el ejemplo 11.1 (ver página 406).

72

73

74

client.stop();//cerrar conexión.

 $}//fin de if(cliente){...} de la línea 28.$

75 $\}//\text{fin de la función loop}()\{...\}$ línea 26.

Serial.println("Cliente desconectado");

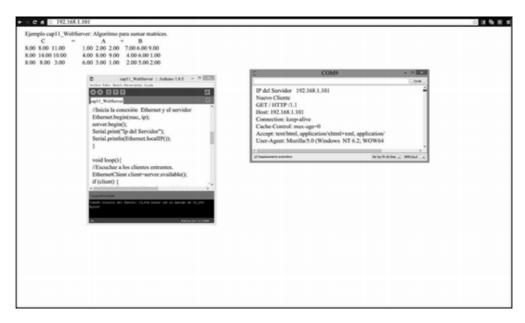


Figura 11.11 Desplegado Web en la dirección IP de la tarjeta Ethernet Shield.

♣ ♣ ♣ Ejemplo 11.3

Desplegar en una página Web la información que proporciona un sensor de voltaje.

Solución

La presentación de información en páginas Web de sensores o detectores de variables físicas es un potencial que tiene el sistema Arduino, en este ejemplo se ilustra la forma de exhibir voltaje que proporciona un potenciómetro conectado a la tarjeta Arduino UNO.

Considere el código Arduino 11.3, el cual describe al sketch **cap11_WebAdq**. En las primeras líneas de este programa se utilizan las librerías de SPI y de Ethernet, así las funciones de Ethernet trabajan correctamente para establecer la configuración y protocolo de comunicación a través de Ethernet; la dirección MAC de la tarjeta Shield (ver figura 11.6b) y dirección IP (cuadro 11.1) se emplean para programar el direccionamiento de enlace, el cual se establece por medio del puerto 80.

De las líneas 7 a la 19 se establece el formulario en HTLM en un arreglo de tipo char para generar los letreros que exhibirá la página Web. La rutina de configuración **setup()** comienza en la línea 20, establece la conexión con la computadora a través del puerto USB e inicializa la comunicación Ethernet.

○ Código Arduino 11.3: sketch cap11_WebAdq

```
Arduino. Aplicaciones en Robótica y Mecatrónica.
  Capítulo 11 Ethernet.
  Fernando Reves Cortés y Jaime Cid Monjaraz.
  Alfaomega Grupo Editor: "Te acerca al conocimiento".
   Sketch cap11_WebAdq.ino
 1 #include <SPI.h>//protocolo: serial peripheral interface.
 2 #include <Ethernet.h>//librería para comunicación Ethernet.
 3 byte mac[] =\{0x90, 0xA2, 0xDA, 0x0D, 0x46, 0xAD\};//dirección MAC.
4 byte ip[] = { 192, 168, 1, 101};//dirección IP de la página Web.
5 float voltaje;//variable para registrar la información del sensor.
 6 EthernetServer server(80);//Ethernet server port 80.
7 \operatorname{const} \operatorname{char} \operatorname{html}[] = "< \operatorname{html}>"
8 "<head>"
9 "<title>Tarjeta Arduino y Ethernet Shield</title>"
10 "</head>"
11 "<body width=100% height=100%>"
12 "<center>"
13 "<h1>Adquisición de datos</h1>"
14 "<form>"
15 "<P>ALFAOMEGA "
17 "<input type=submit value=Repite name=2 style=width:150px;height:100px;background-color:#00ff00'>"
19 "</html>"; //formulario en código HTLM para generar los letreros de la página Web.
20 void setup(){ //rutina de configuración.
      Serial.begin(9600);//comunicación por puerto USB a la computadora.
      Ethernet.begin(mac, ip);//dirección MAC e IP de la tarjeta Ethernet Shield.
22
23
      server.begin();//inicializa protocolo de comunicación Ethernet.
24 }
25 //Rutina principal de programación.
26 void loop() {
27
      char c;
      EthernetClient client = server.available();//escuchar clientes entrantes.
```

```
Continúa código Arduino 11.3a: sketch cap11_WebAdq
  Arduino. Aplicaciones en Robótica y Mecatrónica.
   Capítulo 11 Ethernet.
   Fernando Reyes Cortés y Jaime Cid Monjaraz.
   Alfaomega Grupo Editor: "Te acerca al conocimiento".
   Continuación del sketch cap11_WebAdq.ino
29
       if (client) {
          server.print(html);//puesta del formato HTML en servidor.
30
          while (client.connected()) {
31
32
             c = client.read();
             if(c=='1')
33
                client.print("Termina Adquisición");
34
                break;
35
             \frac{1}{\sin de} if(c=='1'){...}.
36
             if (client.available()) {
37
                voltaje=5.0*analogRead(A3)/1023.0;
38
                client.print(" ");
39
                client.println(voltaje);
40
                if(c=='2') client.print("Repite Adquisición");
41
                if(c=='3') client.print("Adquiere Datos");
42
             }//  fin de if (client.available()) \{...\}.
43
             client.stop();//cierra conexión.
44
          }//fin de while (client.connected()) {...}.
45
       \}// fin de if(client)\{...\}.
46
47
       delay(10); // da tiempo al explorador web para recibir datos.
```

Dentro de la rutina **loop()** (línea 28) se encuentra la programación principal para desplegar la información del sensor en la página Web. Los dispositivos que se conectan a esa página Web (clientes) son detectados o "escuchados" a través de la función **server.available()**; como se indica en la línea 28. Dicha función retorna un valor booleano, el cual se emplea en la línea 29, si este valor es verdadero, entonces envía el formulario HTLM a la página Web (línea 30), el cual establece las opciones de menú e interface para el usuario, quien selecciona botones sobre el formulario para enviar a la tarjeta Arduino el tipo de respuesta; esta información se lee en la línea 32 a través de la función **client.read()**.

48 }//para ejecutar este sketch, consular la página 406.

La adquisición de datos para leer el voltaje del potenciómetro se lleva a cabo en la línea 38, obteniendo lecturas normalizadas entre 0 V a 5 V, es decir: $\frac{5}{1023}$ analogRead(A3) V; este rango se predefine dentro del intervalo de lectura que puede realizar la tarjeta Arduino, ya que las terminales del potenciómetro se encuentran conectadas a 5 V y a tierra (GND). Mientras que el cursor o terminal central del potenciómetro está conectado al canal analógico A3 como se indica en la figura 11.12. El desplegado de esa información se envía para su presentación a la página Web en la línea 40; la figura 11.13 muestra la página Web generada desde la tarjeta Arduino por medio de la interface Ethernet Shield. El procedimiento para ejecutar



Figura 11.12 Sensor de voltaje conectado a la tarjeta Ethernet Shield.

correctamente el sketch cap11_WebAdq se describe en la página 406.



Figura 11.13 Página Web para adquisición de datos por Ethernet.

11.7 Resumen 415

11.7 Resumen



E THERNET es un estándar de redes de área local definiendo las características de cableado y protocolo de comunicación. El sistema Arduino puede expandir sus características potenciales en la red por medio de la tarjeta Ethernet Shield y mediante la librería Ethernet, la cual contiene las funciones necesarias para comunicase a servidores utilizando el protocolo TCP/IP. Para tal efecto, se requieren conocer las direcciones IP y MAC, datos fundamentales en la programación de la tarjeta Ethernet Shield.

IP es una dirección que proporciona conectividad a la red TCP/IP; es una etiqueta numérica que identifica a la tarjeta de la red de un dispositivo como computadora, tablet, iphone, etc. La dirección IP está compuesta por cuatro cifras dentro del rango de 0 a 255, cada cifra está separada por un punto. Cada tarjeta tiene una dirección IP exclusiva. Otro dato necesario para conectarse a la red por medio de la tarjeta Ethernet Shield es la dirección MAC, la cual está formada por un conjunto de 48 bits (12 caracteres hexadecimales) para identificar a esta tarjeta de manera única. Este identificador es proporcionado por el fabricante de la tarjeta.

Para que la tarjeta Ethernet Shield pueda conectarse a la red es necesario configurarla a través de las direcciones IP y MAC. La dirección IP es un arreglo de cuatro elementos de tipo entero, mientras que MAC es un arreglo tipo byte de seis elementos en formato hexadecimal. Dentro del código de la subrutina **setup()** se realiza el siguiente procedimiento:

Debido a que la red TCP/IP tienen una arquitectura llamada "cliente-servidor", la tarjeta Arduino puede configurarse como cliente o servidor. Para el caso de la configuración servidor, es necesario declarar un objeto del tipo **EthernetServer**, por ejemplo: **EthernetServer** robotserver(puerto); donde puerto es el enlace usado por el servidor para escuchar las solicitudes o peticiones de los clientes que desean conectarse al servidor. Posteriormente hay que ponerlo en escucha por medio de la función robotservidor.begin(); esta función está colocada dentro de la subrutina de configuración setup(). Arduino también tiene el potencial de conectarse a la

red a cualquier dispositivo de red que actuará como servidor, de esta forma la tarjeta Ethernet Shield puede solicitar recursos externos, para esto se utiliza la creación de objetos, por ejemplo **EhernetClient Robotcliente**; posteriormente realizar la conexión al servidor por medio de la función: **Robotcliente.connect()**; cuya IP se haya especificado en **Ethernet.begin(ip,max)**.



11.8 Referencias selectas

E THERNET es una tecnología estratégica y de vanguardia que influye notablemente en nuestras vidas y se emplea de manera cotidiana. A continuación se recomienda al lector consultar las siguientes referencias.

Para desarrollar pequeñas redes de Ethernet, información de Arduino Ethernet Shield y descripción del cable categoría 5 UTP, puede ir a los siguientes enlaces:

- arduino.cc/en/Guide/ArduinoEthernetShield
- web support.apple.com/kb/HT2274
- support.apple.com/kb/HT1433

Para desarrollar páginas Web:

- MEDIAactive. "WEB: Aprender a crear su primera web". Alfaomega-Marcombo, 2013.
- Alian Pipes. "Diseño de sitios Web". Promopress, 2011.

Alfaomega

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

FERNANDO REYES CORTÉS • JAIME CID MONJARAZ

11.9 Problemas propuestos



Dentro de las tecnologías más utilizadas en todo el mundo, sin duda es la comunicación por Ethernet. A continuación se propone un conjunto de problemas para mejorar y reforzar los conocimientos adquiridos por el lector en este capítulo.

- 11.9.1 Diseñar un sketch que despliegue en una página Web el siguiente mensaje: "Sistema Arduino comunicándose con el mundo a través de Ethernet". Utilice cualquiera de los modelos de tarjeta Arduino que acepten a Ethernet Shield.
- 11.9.2 Considere las siguientes matrices:

$$A = \begin{bmatrix} 1 & 3.3 & 6.5 & 4.8 \\ 9.2 & 3.33 & 3.1416 & 2.289 \\ 5.678 & 2.13 & 8.12 & 9 \\ 8 & 1.23 & 9.01 & 2.51 \end{bmatrix} \qquad B = \begin{bmatrix} 3 & 8 & 9.5 & 8.1 \\ 2.4 & 6.7 & 7.12 & 14.67 \\ 31.56 & 8.11 & 9.25 & 7.18 \\ 9.45 & 23.13 & 5.45 & 28.67 \end{bmatrix}$$

Implementar en lenguaje C los siguientes algoritmos sobre operaciones aritméticas de matrices:

- a) Suma.
- b) Resta.
- c) Multiplicación.
- d) Invertir matriz.

El resultado de la correspondiente operación presentarlo en una página Web.

- 11.9.3 Conecte a la tarjeta Arduino (UNO, Mega, Leonardo o Due) una celda solar, despliegue el sensado de temperatura en una página Web.
- 11.9.4 Conecte en la tarjeta Ethernet Shield un diodo LED como se indica en la figura 11.14a, diseñe un skecth que genere una página Web (figura 11.14b) para apagar o encender la emisión de luz del LED.
- 11.9.5 Considere un diodo LED del tipo RGB conectado a la tarjeta Ethernet Shield como se indica en la figura 11.15a, diseñar un sketch para generar una página Web por una tarjeta Arduino compatible con Ethernet Shield donde pueda controlar los colores del diodo LED RGB, la figura 11.15b sugiere la forma del menú para acceder a los colores RGB.

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

Fernando Reyes Cortés • Jaime Cid Monjaraz

Alfaomega





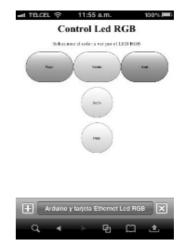
(a) Conexión del LED.

(b) Página Web del LED.

Figura 11.14 Control de encendido y apagado del diodo LED.



(a) Conexión del LED RGB.



(b) Página Web LED RGB.

Figura 11.15 Circuito y página Web para controlar los colores del LED RGB.

Alfaomega

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

FERNANDO REYES CORTÉS • JAIME CID MONJARAZ



Manejo de interrupciones

Capítulo

Capítulo Web

- 12.1 Introducción
- 12.2 Tipos de interrupciones
- 12.3 Rutinas de servicio de interrupciones
- 12.4 Aplicaciones de control en tiempo real
- 12.5 Resumen
- 12.6 Referencias selectas
- 12.7 Problemas propuestos

Competencias

Presentar la descripción y manejo de interrupciones del sistema Arduino, así como sus aplicaciones en control de procesos en tiempo real.

Desarrollar habilidades en:

Descripción de las interrupciones del sistema Arduino.

Tipos de interrupciones.

Subrutinas de servicio de interrupciones.

Control de procesos en tiempo real.

Aplicaciones en control automático.

Capítulo

Capítulo Web

- 13.1 Introducción
- 13.2 WiFi
- 13.3 Puntos de acceso
- 13.4 WiFi Shield
- 13.5 Resumen
- 13.6 Referencias selectas
- 13.7 Problemas propuestos

Competencias

Presentar los conceptos fundamentales de la comunicación WiFi, estándares y modos de operación

Desarrollar habilidades en:

Descripción de la tecnología WiFi.

Estándares para WiFi.

Modo de operación.

Acceso remoto.

WiFi Shield.

Ejemplos y aplicaciones.

LabVIEW

Capítulo

Capítulo Web

- 14.1 Introducción
- 14.2 Ambiente de programación LabVIEW
- 14.3 Programación LabVIEW
- 14.4 Adquisición y desplegado de datos
- 14.5 Resumen
- 14.6 Referencias selectas
- 14.7 Problemas propuestos

Competencias

Presentar la descripción del entorno de programación LabVIEW y su comunicación con las tarjetas electrónicas del sistema Arduino.

Desarrollar habilidades en:

Introducción.

Ambiente de programación LabVIEW.

Adquisición y desplegado de datos.

Comunicación con las tarjetas Arduino

Control de procesos.

Aplicaciones.

MATLAB, 3, 16

Llaves, 58

 $\{...\}, 58$

/*...*/, 58

//, 58

?, 103

; , 58



ambiente de programación Arduino, 24

amortiguamiento crítico, 283

analogRefference, 152

analogWrite, 149

analogWrite, 149

apuntadores, 124

ANSI, 55

Arduino, 3, 14

Diecimila, 11

Due, 7

Duemilanove, 11

Ethernet, 11

Galileo, 7, 11

Leonardo, 11

Mega, 11

Mini, 11

UNO, 11

Wi-Fi, 11

Yune, 11

ARM, 7

armadura, 208

arquitectura abierta, 3, 12

arreglos, 85

 $\operatorname{arreglos}$

unidimensionales, 85

arreglos

bidimensionales, 86

 $\operatorname{arreglos}$

multidimensionales, 87

arrays, 66

ATmega, 4

ATMEL, 3, 8, 12

AVR-RISC, 4



boolean, 64

byte, 64

break, 119

Bluetooh, 3, 333

Bluetooth, 338

Bluetooth(...), 353



char, 64

client.connect, 398

client.write, 398

conmutador, 208

const, 68

constantes

matemáticas, 135

continue, 119

estator, 208 Hernando Barragán, 14 constantes del sistema Arduino, 137 estructuras, 124 HIGH, 74 constraint, 168 Ethernet, 3, 377 control, 281 Ethernet Shield, 391 control de Ethernet temperatura, 314 IDE, 19 Ethernet.begin, 394 $\cos, 171$ Instrucciones Ethernet.LocalIP, 394 de programación, 94 Creative Commons, 5 EthernetServer, 395 Instrrucciones condicionales, 94 if, 104, 108 David Cuartielles, 14 if-else, 98 firma no reconocida, 22 David Mellis, 14 int, 65 float, 67 default, 103 Integración numérica, 256 for, 106 delay, 166 IVREA, 14 fopen, 355 delayMicroseconds, 166 identificadores, 62 fscanf, 356 digitalRead, 152 INPUT, 74 fwrite, 355 Diferenciación numérica, 262 functiones, 87 digitalWrite, 142 $do\{...\}$ while(), 115 LabVIEW, 3, 16 double, 66 Lenguaje C, 53 Gianluca Martino, 14 drivers Arduino, 21 librerías, 93 goto, 94, 120 librerías \mathbf{H} allocate.h, 131 math.h,

Alfaomega

EEPROM, 7

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

FERNANDO REYES CORTÉS • JAIME CID MONJARAZ

131, 135, 136

Hercules, 350

| stdio.h, 131, 132 | menu Editor, 31 | $\{\}, 58$ |
|---|--|--------------------------|
| stdlib.h, 131 , 132 | micros, 165 | /**/, 58 |
| string, 131 | millis, 165 | //, 58 |
| Ethernet, 131, 394 | min, 166 | ; , 58 |
| Servo.h, 131 | Modificadores | ?, 103 |
| SoftwareSerial, 342 | estándar, 70 | OUTPUT, 74 |
| Stepper.h, 131 | Módulo Bluetooth | |
| Wifi.h, 131 | JY-MCU, 344 | Р |
| Librerías y | motores a | |
| funciones Arduino, 127 | pasos, 210 | palabras reservadas, 63 |
| LM35, 321 | motor 28BYJ-48, 218 | - |
| $\log, 66$ | motores a pasos | PID, 315 |
| loop, 138 | unipolares, 215 | pinMode, 141 |
| Lorentz, 208 | motores a pasos | portabilidad, 54 |
| LOW, 74 | bipolares, 222 | pow, 170 |
| | Motores de corriente directa, 194 | proporcional |
| \mathbf{M} | | derivativo-integral, 315 |
| | Motor Shield, 197 | pullup, 141 |
| MATLAB , 245 | О | pulsIn, 176 |
| malloc, 133 | | PWM, 7 |
| magneto | open hardware, 14 | \mathbb{R} |
| permanente, 214 | Operadores, 76 | |
| Manipulación de | aritméticos, 77 | |
| bits, 145 | lógicos, 84 | RAM, 7 |
| $\mathrm{map},168$ | nivel de bits, 84 | random, 171 |
| Massimo Banzi, 14, 15 | relacionales, 83 | randomSeed, 172 |
| max, 167 | comparación, 83 | reglas de |
| mecatrónica, 207 | Llaves, 58 | sintonía, 317 |
| HINO ADLICACIONES EN PODÓTICA Y MEGATRÓNICA | EDDNANDO REVES CODTÉS A TAIME CID MONIADAZ ALEAGMECA | |

Fernando Reyes Cortés • Jaime Cid Monjaraz

Alfaomega

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

signed, 68 Trama de respuesta oscilatoria, 283 Ethernet, 387 $\sin, 170$ respuesta sistema sobreamortiguada, 284 empotrado Arduino, 7 respuesta Sketch, 9 subamortiguada, 283 uniones, 124 sketchs, 9 return, 91 unsigned, 68 Sketchbook, 27 unsigned char, 64 short, 65 unsigned int, 65 sqrt, 170 step, 237 unsigned long, 66 Serial, 179 Stepper, 236 USB, 3 Serial.available, 180 sistema discreto, 285 unsigned short, 65 Serial.begin, 179 string, 65 Serial.end, 179 sistema de Serial.print, 183 segundo orden, 282 Serial.println, 185 subrutinas, 87 variables, 61 Serial.read, 180 switch, 102 variable Serial.write, 185 fase, 281 servoamplificador, 206 variable Servos, 191 de estado, 281 servo.attached, 207 tan, 171 variables de estado, 284 servo.detach, 207 Tecnología Ethernet, 380 servo.read, 208 termopar, 320 servo.write, 208 Tipos de datos, 64 setup, 138 while, 123 tiempo real, 5 sizeof, 139 Wi-Fi, 3 Tom Igoe, 14 sistema empotrado, 6 tone, 174 word, 65

Alfaomega

ARDUINO. APLICACIONES EN ROBÓTICA Y MECATRÓNICA

Fernando Reyes Cortés • Jaime Cid Monjaraz